

Parallel Engineering and Scientific Subroutine Library  
for AIX, Version 3 Release 2, and  
Parallel Engineering and Scientific Subroutine Library  
for Linux on POWER, Version 3 Release 2



# Guide and Reference



Parallel Engineering and Scientific Subroutine Library  
for AIX, Version 3 Release 2, and  
Parallel Engineering and Scientific Subroutine Library  
for Linux on POWER, Version 3 Release 2



# Guide and Reference

**Note:**

Before using this information and the product it supports, read the information in "Notices" on page 987.

**Fourth Edition (April 2005)**

This edition applies to:

- Version 3 Release 2 of the IBM Parallel Engineering and Scientific Subroutine Library (Parallel ESSL) for Advanced Interactive Executive (AIX) licensed program, program number 5765-F84
- Version 3 Release 2 of the IBM Parallel Engineering and Scientific Subroutine Library (Parallel ESSL) for Linux on POWER licensed program, program number 5765-G18

and to all subsequent releases and modifications until otherwise indicated by new editions.

In this document, Parallel ESSL refers to both of the above products (unless a differentiation between Parallel ESSL for AIX and Parallel ESSL for Linux is explicitly specified).

Changes are periodically made to the information herein. Significant changes or additions to the text and illustrations are marked by a vertical line (|) to the left of the change.

Order IBM publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

IBM welcomes your comments. Address your comments as follows:

- World Wide Web: [http://www-1.ibm.com/servers/eserver/pseries/library/sp\\_books/feedback.html](http://www-1.ibm.com/servers/eserver/pseries/library/sp_books/feedback.html)
- Mail:
  - International Business Machines Corporation
  - Department 55JA, Mail Station P384
  - 2455 South Road
  - Poughkeepsie, NY 12601-5400
  - United States of America
- FAX:
  - (United States & Canada): 1+845+432-9405
  - (Other countries): Your International Access Code +1+845+432-9405
- IBMLink™ (United States customers only): IBMUSM(MHVRCFS)
- IBM Mail Exchange: USIB6TC9 at IBMAIL
- Internet e-mail: [mhvrcfs@us.ibm.com](mailto:mhvrcfs@us.ibm.com)

If you would like a reply, be sure to include the following in your comment or note:

- Your name, address, telephone number, or FAX number
- Title and order number of this book
- Page number or topic related to your comment

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1995, 2005. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.



---

# Contents

<b>About This Book . . . . .</b>	<b>vii</b>
How to Use This Book . . . . .	vii
How to Find a Subroutine Description . . . . .	viii
Where to Find Related Publications . . . . .	viii
How to Look Up a Bibliography Reference . . . . .	viii
Special Terms . . . . .	ix
How to Interpret Product Names Used in This Document . . . . .	x
Abbreviated Names . . . . .	x
Fonts. . . . .	xi
Scalar Data Notations . . . . .	xi
Special Characters, Symbols, Expressions, and Abbreviations . . . . .	xii
Interpreting the Subroutine Descriptions . . . . .	xiii
Syntax . . . . .	xiii
On Entry . . . . .	xiv
On Return. . . . .	xv
Notes and Coding Rules . . . . .	xv
Error Conditions . . . . .	xv
Example . . . . .	xv
<b>Summary of Changes . . . . .</b>	<b>xvii</b>

---

## Part 1. Guide Information . . . . . 1

<b>Chapter 1. Overview, Requirements, and List of Subroutines . . . . .</b>	<b>3</b>
Overview of Parallel ESSL. . . . .	3
How Parallel ESSL Works . . . . .	4
Accuracy of the Computations . . . . .	5
The Fortran Language Interface to the Parallel ESSL Subroutines. . . . .	5
Hardware and Software Products That Can Be Used with Parallel ESSL . . . . .	5
Hardware Products Supported by Parallel ESSL . . . . .	6
Operating Systems Supported by Parallel ESSL . . . . .	6
Software Products Required by Parallel ESSL . . . . .	7
Thread Safety and Parallel ESSL . . . . .	10
Installation and Customization of Parallel ESSL . . . . .	10
Software Products Required for Displaying Parallel ESSL Documentation . . . . .	11
Parallel ESSL Internet Resources . . . . .	11
Getting on the ESSL Mailing List . . . . .	12
BLACS—Usage in Parallel ESSL for Communication . . . . .	12
List of Parallel ESSL Subroutines . . . . .	13
Level 2 PBLAS . . . . .	13
Level 3 PBLAS . . . . .	13
Linear Algebraic Equations . . . . .	14
Eigensystem Analysis and Singular Value Analysis . . . . .	17
Fourier Transforms . . . . .	17
Random Number Generation . . . . .	18
Utilities. . . . .	18

<b>Chapter 2. Distributing Your Data . . . . .</b>	<b>21</b>
Concepts . . . . .	21
About Global Data Structures . . . . .	21
About Process Grids . . . . .	21
What to Do in Your Program . . . . .	22
Block, Cyclic, and Block-Cyclic Data Distributions . . . . .	22
Specifying and Distributing Data in Your Program . . . . .	26
Specifying Block-Cyclically-Distributed Vectors and Matrices . . . . .	26
Specifying Block-Cyclically-Distributed Matrices for the Banded Linear Algebraic Equations . . . . .	29
Distributing Data Structures . . . . .	33
Vectors . . . . .	33
Matrices . . . . .	40
Specifying Sparse Matrices for the Fortran 90 and Fortran 77 Sparse Linear Algebraic Equations . . . . .	58
Specifying Sequences for the Fourier Transforms . . . . .	64

<b>Chapter 3. Coding and Running Your Program . . . . .</b>	<b>77</b>
Coding Tips for Optimizing Parallel Performance. . . . .	77
Choosing How Many MPI Tasks and Computational Threads to Use . . . . .	77
Parallel ESSL Techniques . . . . .	77
Avoiding Conflicts with Parallel ESSL and ESSL Routine Names . . . . .	79
Coding Your Program . . . . .	79
Using the BLACS . . . . .	80
Using Extrinsic Procedures—The Fortran 90 Sparse Linear Algebraic Equation Subroutines. . . . .	86
Setting Up the Parallel ESSL Header File for C and C++ . . . . .	87
Setting Up the C Interface for the BLACS Header File for C and C++ . . . . .	87
Application Program Outline . . . . .	87
Application Program Outline for the Fortran 90 Sparse Linear Algebraic Equations and Their Utilities. . . . .	89
Application Program Outline for the Fortran 77 Sparse Linear Algebraic Equations and Their Utilities. . . . .	90
Running Your Program . . . . .	91
Running Your Program on AIX. . . . .	91
Running Your Program on Linux . . . . .	94

<b>Chapter 4. Migrating Your Programs . . . . .</b>	<b>99</b>
Migrating to Parallel ESSL for AIX Version 3 Release 2 . . . . .	99
Migrating to Parallel ESSL for Linux Version 3 Release 2 . . . . .	99
Migrating to Parallel ESSL Version 3 Release 1. . . . .	99
Migrating from ScaLAPACK to Parallel ESSL . . . . .	99
Migrating from ScaLAPACK 1.5 to Parallel ESSL . . . . .	99

<b>Chapter 5. Using Error Handling . . . . .</b>	<b>101</b>
--	------------

Where to Find More Information About Errors . . .	101
Getting Help from IBM Support . . . . .	101
National Language Support . . . . .	102
PESSL_ERROR_SYNC Environment Variable . . .	103
Dealing with Errors . . . . .	103
Program Exceptions . . . . .	104
Input-Argument Errors . . . . .	104
Computational Errors . . . . .	105
Resource Errors . . . . .	105
Communication Errors . . . . .	106
Informational and Attention Messages . . .	106
Miscellaneous Errors . . . . .	106
ESSL Error Messages . . . . .	107
MPI Error Messages . . . . .	107
Messages . . . . .	107
Message Conventions . . . . .	107
Input-Argument Error Messages (001-299) .	108
Computational Error Messages (300-399) .	118
Resource Error Messages (400-499) . . .	120
Communication Error Messages (500-599) .	120
Informational and Attention Messages (600-699)	120
Miscellaneous Error Messages (700-799) .	121
Input-Argument Error Messages (800-999) .	122

## Part 2. Reference Information . . . 127

### Chapter 6. Level 2 PBLAS . . . . . 129

Overview of the Level 2 PBLAS Subroutines . .	129
Level 2 PBLAS Subroutines. . . . .	130
PDGEMV and PZGEMV — Matrix-Vector Product for a General Matrix or Its Transpose . . . .	131
PDSYMV and PZHEMV — Matrix-Vector Product for a Real Symmetric or a Complex Hermitian Matrix. . . . .	154
PDGER, PZGERC, and PZGERU — Rank-One Update of a General Matrix . . . . .	168
PDSYR and PZHER — Rank-One Update of a Real Symmetric or a Complex Hermitian Matrix . .	186
PDSYR2 and PZHER2 — Rank-Two Update of a Real Symmetric or a Complex Hermitian Matrix.	197
PDTRMV and PZTRMV — Matrix-Vector Product for a Triangular Matrix or Its Transpose . . .	212
PDTRSV and PZTRSV — Solution of Triangular System of Equations with a Single Right-Hand Side	224

### Chapter 7. Level 3 PBLAS . . . . . 237

Overview of the Level 3 PBLAS Subroutines . .	237
Level 3 PBLAS Subroutines. . . . .	238
PDGEMM and PZGEMM — Matrix-Matrix Product for a General Matrix, Its Transpose, or Its Conjugate Transpose . . . . .	239
PDSYMM, PZSYMM, and PZHEMM — Matrix-Matrix Product Where One Matrix is Real or Complex Symmetric or Complex Hermitian .	256
PDTRMM and PZTRMM — Triangular Matrix-Matrix Product . . . . .	276
PDTRSM and PZTRSM — Solution of Triangular System of Equations with Multiple Right-Hand Sides . . . . .	288

PDSYRK, PZSYRK, and PZHERK — Rank-K Update of a Real or Complex Symmetric or a Complex Hermitian Matrix. . . . .	301
PDSYR2K, PZSYR2K, and PZHER2K — Rank-2K Update of a Real or Complex Symmetric or a Complex Hermitian Matrix. . . . .	316
PDTRAN, PZTRANC, and PZTRANU — Matrix Transpose for a General Matrix . . . . .	336

## Chapter 8. Linear Algebraic Equations 351

Overview of the Dense Linear Algebraic Equation Subroutines . . . . .	351
Overview of the Banded Linear Algebraic Equation Subroutines . . . . .	352
Overview of the Fortran 90 Sparse Linear Algebraic Equation Subroutines. . . . .	352
Overview of the Fortran 77 Sparse Linear Algebraic Equation Subroutines. . . . .	353
Dense Linear Algebraic Equation Subroutines . .	354
PDGESV and PZGESV — General Matrix Factorization and Solve . . . . .	355
PDGETRF and PZGETRF — General Matrix Factorization . . . . .	370
PDGETRS and PZGETRS — General Matrix Solve	381
PDGETRI and PZGETRI — General Matrix Inverse	393
PDGECON and PZGECON — Estimate the Reciprocal of the Condition Number of a General Matrix. . . . .	402
PDGEQRF and PZGEQRF — General Matrix QR Factorization. . . . .	411
PDGELS and PZGELS — General Matrix Least Squares Solution . . . . .	421
PDPOSV and PZPOSV — Positive Definite Real Symmetric or Complex Hermitian Matrix Factorization and Solve . . . . .	435
PDPOTRF and PZPOTRF — Positive Definite Real Symmetric or Complex Hermitian Matrix Factorization . . . . .	449
PDPOTRS and PZPOTRS — Positive Definite Real Symmetric or Complex Hermitian Matrix Solve .	458
PDPOTRI and PZPOTRI — Positive Definite Real Symmetric or Complex Hermitian Matrix Inverse	469
PDPOCON and PZPOCON — Estimation of the Reciprocal of the Condition Number of a Positive Definite Real Symmetric or Complex Hermitian Matrix. . . . .	476
Banded Linear Algebraic Equation Subroutines .	485
PDPBSV — Positive Definite Symmetric Band Matrix Factorization and Solve . . . . .	486
PDPBTRF — Positive Definite Symmetric Band Matrix Factorization . . . . .	498
PDPBTRS — Positive Definite Symmetric Band Matrix Solve. . . . .	507
PDGTSV and PDDTSV — General Tridiagonal Matrix Factorization and Solve . . . . .	518
PDGTTRF and PDDTTRF — General Tridiagonal Matrix Factorization . . . . .	532
PDGTTRS and PDDTTRS — General Tridiagonal Matrix Solve. . . . .	548
PDPTSV — Positive Definite Symmetric Tridiagonal Matrix Factorization and Solve . .	565

PDPTTRF — Positive Definite Symmetric Tridiagonal Matrix Factorization . . . . .	579
PDPTTRS — Positive Definite Symmetric Tridiagonal Matrix Solve . . . . .	591
Fortran 90 Sparse Linear Algebraic Equation Subroutines and Their Utility Subroutines . . . .	606
PADALL — Allocates Space for an Array Descriptor for a General Sparse Matrix . . . .	607
PSPALL — Allocates Space for a General Sparse Matrix. . . . .	609
PGEALL — Allocates Space for a Dense Vector . .	611
PSPINS — Inserts Local Data into a General Sparse Matrix. . . . .	613
PGEINS — Inserts Local Data into a Dense Vector	617
PSPASB — Assembles a General Sparse Matrix . .	619
PGEASB — Assembles a Dense Vector . . . . .	622
PSPGPR — Preconditioner for a General Sparse Matrix. . . . .	624
PSPGIS — Iterative Linear System Solver for a General Sparse Matrix . . . . .	627
PGEFREE — Deallocates Space for a Dense Vector	632
PSPFREE — Deallocates Space for a General Sparse Matrix. . . . .	633
PADFREE — Deallocates Space for an Array Descriptor for a General Sparse Matrix . . . .	635
Example—Using the Fortran 90 Sparse Subroutines	636
Fortran 77 Sparse Linear Algebraic Equation Subroutines and Their Utility Subroutines . . . .	642
PADINIT — Initializes an Array Descriptor for a General Sparse Matrix . . . . .	643
PDSPINIT — Initializes a General Sparse Matrix	645
PDSPINS — Inserts Local Data into a General Sparse Matrix . . . . .	647
PDGEINS — Inserts Local Data into a Dense Vector. . . . .	652
PDSPASB — Assembles a General Sparse Matrix	655
PDGEASB — Assembles a Dense Vector . . . . .	659
PDSPGPR — Preconditioner for a General Sparse Matrix. . . . .	661
PDSPGIS — Iterative Linear System Solver for a General Sparse Matrix . . . . .	664
Example—Using the Fortran 77 Sparse Subroutines	669

## Chapter 9. Eigensystem Analysis and Singular Value Analysis . . . . . 675

Overview of the Eigensystem Analysis and Singular Value Analysis Subroutines. . . . .	675
Eigensystem Analysis and Singular Value Analysis Subroutines . . . . .	676
PDSYEVX and PZHEEVX — Selected Eigenvalues and, Optionally, the Eigenvectors of a Real Symmetric or Complex Hermitian Matrix . . . .	677
PDSYGVX and PZHEGVX — Selected Eigenvalues and, Optionally, the Eigenvectors of a Real Symmetric or Complex Hermitian Positive Definite Generalized Eigenproblem . . . . .	698
PDSYTRD and PZHETRD — Reduce a Real Symmetric or Complex Hermitian Matrix to Tridiagonal Form . . . . .	724

PDSYGST and PZHEGST — Reduce a Real Symmetric or Complex Hermitian Positive Definite Generalized Eigenproblem to Standard Form . . .	739
PDGEHRD — Reduce a General Matrix to Upper Hessenberg Form . . . . .	753
PDGEBRD and PZGEBRD — Reduce a General Matrix to Bidiagonal Form . . . . .	762
PDGESVD and PZGESVD — Singular Value Decomposition of a General Matrix . . . . .	780

## Chapter 10. Fourier Transforms . . . 797

Overview of the Fourier Transforms Subroutines	797
Acceptable Lengths for the Transforms . . . .	797
Fourier Transforms Subroutines . . . . .	799
PSCFT2 and PDCFT2 — Complex Fourier Transforms in Two Dimensions . . . . .	800
PSRCFT2 and PDRCFT2 — Real-to-Complex Fourier Transforms in Two Dimensions. . . . .	807
PSCRFT2 and PDCRFT2 — Complex-to-Real Fourier Transforms in Two Dimensions. . . . .	812
PSCFT3 and PDCFT3 — Complex Fourier Transforms in Three Dimensions . . . . .	817
PSRCFT3 and PDRCFT3 — Real-to-Complex Fourier Transforms in Three Dimensions . . . .	825
PSCRFT3 and PDCRFT3 — Complex-to-Real Fourier Transforms in Three Dimensions . . . .	831

## Chapter 11. Random Number Generation . . . . . 837

Overview of the Random Number Generation Subroutine . . . . .	837
Random Number Generation Subroutine . . . .	838
PDURNG — Uniform Random Number Generator	839

## Chapter 12. Utilities. . . . . 845

Overview of the Utility Subroutines . . . . .	845
Utility Subroutines . . . . .	846
IPESSL — Determine the Level of Parallel ESSL Installed on Your System . . . . .	847
DESCINIT — Initialize a Type-1 Array Descriptor with Error Checking . . . . .	849
DESCSET — Initialize a Type-1 Array Descriptor	852
ICEIL — Compute the Ceiling of the Division of Two Integers . . . . .	855
ILCM — Compute the Least Common Multiple of Two Positive Integers. . . . .	856
INDXG2L — Compute the Local Row or Column Index of a Global Element of a Block-Cyclically Distributed Matrix. . . . .	857
INDXG2P — Compute the Process Row or Column Index of a Global Element of a Block-Cyclically Distributed Matrix. . . . .	859
INDXL2G — Compute the Global Row or Column Index of a Local Element of a Block-Cyclically Distributed Matrix. . . . .	861
INFOG1L — Compute the Starting Local Row or Column Index and Process Row or Column Index of a Global Element of a Block-Cyclically Distributed Matrix. . . . .	863

INFOG2L — Compute the Starting Local Row and Column Indices and the Process Row and Column Indices of a Global Element of a Block-Cyclically Distributed Matrix. . . . .	865
NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process . . . . .	868
PDLANGE and PZLANGE — General Matrix Norm . . . . .	872
PDLANSY, PZLANSY, and PZLANHE — Real Symmetric, Complex Symmetric, or Complex Hermitian Matrix Norm. . . . .	879

## **Part 3. Appendixes . . . . . 889**

### **Appendix A. BLACS Quick Reference Guide. . . . . 891**

Calling sequences . . . . .	891
Fortran interface for the BLACS . . . . .	891
C interface for the BLACS . . . . .	893
Argument data types. . . . .	894
Argument options. . . . .	894

### **Appendix B. Sample Programs . . . . 895**

Sample Programs and Utilities Provided with Parallel ESSL . . . . .	895
Sample Thermal Diffusion Program . . . . .	896
Thermal Diffusion Discussion Paper. . . . .	898
Program Main . . . . .	902
Module Parameters . . . . .	907
Module Diffusion . . . . .	908
Module Fourier. . . . .	911
Module Scale . . . . .	919
Input Data . . . . .	928

Output Data. . . . .	928
Sample Sparse Linear Algebraic Equations Programs. . . . .	931
Fortran 90 Sample Sparse Program . . . . .	932
Fortran 77 Sample Sparse Program . . . . .	941
Fortran 90 Sample Sparse Program (using the Harwell-Boeing exchange format) . . . . .	950
Sample PARTS Subroutine . . . . .	956
The READ_MAT Subroutine . . . . .	959
The MAT_DIST Subroutine. . . . .	962
The DESYM Subroutine . . . . .	966
Sample Makefiles and Run Script for AIX . . . . .	967
Makefiles and Run Script for Use with SMP Libraries on AIX . . . . .	967
Makefiles and Run Script for Use with GM Libraries on AIX . . . . .	973
Sample Makefiles and Run Script for Linux . . . . .	980
32-Bit Makefile for Linux . . . . .	980
64-Bit Makefile for Linux . . . . .	983
Run Script for Linux . . . . .	985

### **Notices . . . . . 987**

Trademarks . . . . .	988
Software Update Protocol . . . . .	989
Programming Interfaces . . . . .	989

### **Glossary . . . . . 991**

### **Bibliography. . . . . 999**

References . . . . .	999
Parallel ESSL Publications. . . . .	1002
Related Publications. . . . .	1002

### **Index . . . . . 1003**

---

## About This Book

The IBM® Parallel Engineering and Scientific Subroutine Library (Parallel ESSL) is a set of high-performance mathematical subroutines.

This book is a guide and reference manual for use in doing application programming in Fortran, C, and C++. It includes:

- An overview of Parallel ESSL and guidance information for coding and running your program, as well as using error handling
- Reference information for coding each subroutine calling sequence

This book is meant to be used in conjunction with the *ESSL Guide and Reference*. Where information is identical between Parallel ESSL and ESSL, such as matrix storage modes, this book references the appropriate section of the *ESSL Guide and Reference*.

This book is written for a wide class of users: scientists, mathematicians, engineers, statisticians, computer scientists, and system programmers. It assumes a basic knowledge of mathematics, Single Program Multiple Data (SPMD) parallel processing concepts and familiarity with Fortran, C, or C++.

---

## How to Use This Book

**Front Matter** consists of the Table of Contents, a subroutine lookup table, special terms and abbreviated names, and other prefatory information. Use these to find or interpret information in the book.

**Part 1, “Guide Information,” on page 1** provides guidance information for using Parallel ESSL.

- **Chapter 1, “Overview, Requirements, and List of Subroutines,” on page 3** gives an overview of Parallel ESSL and lists required hardware and software products. Read this chapter first to determine the aspects of Parallel ESSL you want to use.
- **Chapter 2, “Distributing Your Data,” on page 21** describes how to distribute your data across processes for various types of data structures: vectors, matrices, and sequences. Use this information when designing and coding your program.
- **Chapter 3, “Coding and Running Your Program,” on page 77** explains coding requirements for calling Parallel ESSL from Fortran, C, and C++ programs, performance coding tips, and how to run your program. Use this information when coding or running your program.
- **Chapter 4, “Migrating Your Programs,” on page 99** describes how to migrate your program to Parallel ESSL. Use this information when updating your program for a new release of Parallel ESSL or when moving from ScaLAPACK to Parallel ESSL.
- **Chapter 5, “Using Error Handling,” on page 101** describes how to use error handling in Parallel ESSL to retrieve information about errors that occur in your program and diagnose problems. Use this information when designing and coding your program as well as diagnosing your problems.

**Part 2, “Reference Information,” on page 127** provides reference information you need to code calling sequences for the Parallel ESSL subroutines. Each chapter



contains an introduction and subroutine descriptions. To understand the information in the subroutine descriptions, see “Interpreting the Subroutine Descriptions” on page xiii. Use the appropriate chapter when coding your program:

- **Chapter 6, “Level 2 PBLAS,” on page 129**
- **Chapter 7, “Level 3 PBLAS,” on page 237**
- **Chapter 8, “Linear Algebraic Equations,” on page 351**
- **Chapter 9, “Eigensystem Analysis and Singular Value Analysis,” on page 675**
- **Chapter 10, “Fourier Transforms,” on page 797**
- **Chapter 11, “Random Number Generation,” on page 837**
- **Chapter 12, “Utilities,” on page 845**

**Appendix A, “BLACS Quick Reference Guide,” on page 891** provides a list of calling sequences for the BLACS subroutines.

**Appendix B, “Sample Programs,” on page 895** contains a sample Fortran 90 application program using Parallel ESSL. It also contains sample application programs using the Fortran 90 and Fortran 77 sparse linear algebraic equation subroutines.

**“Glossary” on page 991** contains definitions of terms used in this book.

**“Bibliography” on page 999** provides information about publications related to Parallel ESSL. Use it to identify and order publications with supporting information.

---

## How to Find a Subroutine Description

If you want to locate a subroutine description and you know the subroutine name, you can find it listed individually or under the entry “subroutines” in the Index.

---

## Where to Find Related Publications

If you have a question about ESSL products, IBM clustered servers, or a related product, the online resources listed in Table 6 on page 11 and in “Related Publications” on page 1002 make it easy to find the information for which you are looking.

In addition, included in “Bibliography” on page 999 is a list of math background publications you may find helpful, along with the necessary information for ordering them from independent sources. See “Bibliography” on page 999.

---

## How to Look Up a Bibliography Reference

Special references are made throughout this book to mathematical background publications and software libraries, available through IBM, publishers, or other companies. All of these are described in detail in the bibliography. A reference to one of these is made by using a number enclosed in square brackets. The number refers to the item listed under that number in the bibliography. For example, reference [1] cites the first item listed in the bibliography.

---

## Special Terms

Standard data processing and mathematical terms are used in this book. Terminology is generally consistent with that used for Fortran. See the Glossary for more definitions of terms used in this book.

**Distribution:** Used to describe the method in which global data structures are divided among processes. Reference reports may use the term **decomposition** to mean the same thing.

**Global:** Used to identify arguments that must have the same value on all processes.

**Local:** Used to identify arguments that may have different values on different processes.

**LOCp():** For block-cyclic data distribution,  $\text{LOCp}(M\_)$  represents the number of rows that a process would receive if  $M\_$  was distributed block-cyclically over  $p$  rows of its process column.

The *ScaLAPACK Users' Guide* uses  $\text{LOCr}$ , which is equivalent to  $\text{LOCp}$ .

**LOCq():**  $\text{LOCq}()$  can be used in three ways:

- For block-cyclic data distribution,  $\text{LOCq}(N\_)$  represents the number of columns that a process would receive if  $N\_$  was distributed block-cyclically over  $q$  columns of its process row.
- For block-column data distribution,  $\text{LOCq}(n)$  represents the number of columns that a process would receive if  $n$  was distributed block over  $q$  processes.
- For block-plane data distribution,  $\text{LOCq}(n)$  represents the number of planes that a process would receive if  $n$  was distributed block over  $q$  processes.

The *ScaLAPACK Users' Guide* uses  $\text{LOCc}$ , which is equivalent to  $\text{LOCq}$ .

**Optional:** Indicates an argument does not have to be coded and is assigned a default value if the argument is not present.

**Process:** Indicates the logical CPUs identified in the process grid. Referenced reports may also use the terms **processor** or **node** to mean the same thing.

**Process Grid:** Indicates a way to view a parallel machine as a logical one- or two-dimensional rectangular grid.

For one-dimensional process grids, the variables  $p$  and  $np$  are used interchangeably to indicate the number of processes in a row or column of the process grid.

For two-dimensional process grids, the variables  $p$  and  $nprow$  are used interchangeably to indicate the number of rows in the process grid. The variables  $q$  and  $npcol$  are used interchangeably to indicate the number of columns in the process grid.

Referenced reports or manuals may also use the terms **processor mesh**, **processor template**, **processor shape**, or **processor grid**. These all mean the same thing.

**Required:** Indicates an argument must be coded in the calling sequence.

**Scope:** Scope can be used in two ways:

1. Refers to the portion of the parallel computer program within which the definition of an argument remains unchanged. When the scope of an argument is defined as global, the argument must have the same value on all processes. When the scope of an argument is defined as local, the argument may have different values on different processes.
2. In Appendix A, “BLACS Quick Reference Guide,” on page 891, scope indicates the processes that participate in the broadcast and global operations. It can equal ‘all’, ‘row’, or ‘column’.

**Short and Long Precision:** Because Parallel ESSL can be used with more than one programming language, the terms **short precision** and **long precision** are used in place of the Fortran terms **single precision** and **double precision**.

**Subroutines and Subprograms:** A **subroutine** is a named sequence of instructions within the Parallel ESSL library, whose execution is invoked by a call. A subroutine can be called in one or more user programs and at one or more times within each program. The Parallel ESSL subroutines are referred to as **subprograms** in the areas of Level 2 and 3 Parallel Basic Linear Algebra Subprograms (PBLAS). The term subprograms is used because it is consistent with the Basic Linear Algebra Subprograms (BLAS).

---

## How to Interpret Product Names Used in This Document

Parallel ESSL refers to the Parallel Engineering and Scientific Subroutine Library product.

ESSL refers to the Engineering and Scientific Subroutine Library product.

MPI refers to the Message Passing Interface provided by Parallel Environment (PE).

---

## Abbreviated Names

The abbreviated names used in this book are defined below.

Short Name	Full Name
AIX®	Advanced Interactive Executive
BLACS	Basic Linear Algebra Communication Subprograms
BLAS	Basic Linear Algebra Subprograms
CSM	Cluster Systems Management
ESSL	Engineering and Scientific Subroutine Library
FDDI	Fiber Distributed Data Interface
HPS	High Performance Switch
HTML	Hypertext Markup Language
IP	Internet Protocol
LAPACK	Linear Algebra Package
LAPI	Low-level Application Programming Interface
MPI	Message Passing Interface
MPL	Message Passing Library



Short Name	Full Name
NLS	National Language Support
PDF	Portable Document Format
PE	Parallel Environment
PBLAS	Parallel Basic Linear Algebra Subprograms
IBM @server Clusters 1600	Highly scalable cluster solution comprised of POWER architecture-based symmetric multiprocessing (SMP) AIX 5L or Linux servers
PSSP	Parallel System Support Programs
ScaLAPACK	Scalable Linear Algebra Package
SMP	Symmetric Multi-Processing
SPMD	Single Program Multiple Data
US	User Space

## Fonts

This book uses a variety of special fonts to distinguish between many mathematical and programming items. These are defined below.

Special Font	Example	Description
Italic with no subscripts	<i>m, incx, uplo</i>	A calling sequence argument or mathematical variable
Italic with subscripts	<i>x<sub>1</sub>, a<sub>ij</sub>, y<sub>k1, k2</sub></i>	An element of a vector, matrix, or sequence
Bold italic lowercase	<b><i>x, y, z</i></b>	A vector or sequence
Bold italic lowercase with subscripts	<b><i>x<sub>ix:ix+n-1</sub></i></b>	A vector, with defined bounds
Bold italic uppercase	<b><i>A, B, C</i></b>	A matrix
Bold italic uppercase with subscripts	<b><i>A<sub>ia:ia+m-1, ja:ja+n-1</sub></i></b> <b><i>X<sub>ix:ix+n-1, ja:ja</sub></i></b>	A submatrix, with defined bounds A vector (a special form of submatrix), with defined limits
Gothic uppercase	A, B, C, AGB NPROW=2	An array A Fortran statement

## Scalar Data Notations

Following are the special notations used in this book for scalar data items. These notations do not imply usage of any precision, short or long.

Data Item	Example	Description
Character item	'T'	Character(s) in single quotation marks
Logical item	.TRUE. .FALSE.	True or false logical value, as indicated
Integer data	1	Number with no decimal point
Real data	1.6	Number with a decimal point
Complex data	(1.0, -2.9)	Real part followed by the imaginary part

## Special Characters, Symbols, Expressions, and Abbreviations

The mathematical and programming notations used in this book are consistent with traditional mathematical and programming usage. These conventions are explained below, along with special abbreviations that are associated with specific values.

Item	Description
Greek letters: $\alpha, \sigma, \omega, \Omega$	Symbolic scalar values
$ a $	The absolute value of $a$
$a \bullet b$	The dot product of $a$ and $b$
$x_i$	The $i$ -th element of vector $x$
$c_{ij}$	The element in matrix $C$ at row $i$ and column $j$
$x_1 \dots x_n$	Elements from $x_1$ to $x_n$
$i = 1, n$	$i$ is assigned the values 1 to $n$
$y \leftarrow x$	Vector $y$ is replaced by vector $x$
$xy$	Vector $x$ times vector $y$
$a^k$	$a$ raised to the $k$ power
$e^x$	Exponential function of $x$
$A^T; x^T$	The transpose of matrix $A$ ; the transpose of vector $x$
$\bar{x}; \bar{A}$	The complex conjugate of vector $x$ ; the complex conjugate of matrix $A$
$\bar{x}_i; \bar{c}_{jk}$	<p>The complex conjugate of the complex vector element <math>x_i</math>, where:  if <math>x_i = (a_i, b_i)</math>,  then <math>\bar{x}_i = (a_i, -b_i)</math></p> <p>The complex conjugate of the complex matrix element <math>c_{jk}</math></p>
$x^H; A^H$	The complex conjugate transpose of vector $x$ ; the complex conjugate transpose of matrix $A$
$I$	Identity matrix
$\sum_{i=1}^n x_i$	The sum of elements $x_1$ to $x_n$
$\sqrt{a+b}$	The square root of $a+b$
$\ x\ _2$	<p>The Euclidean norm of vector <math>x</math>, defined as:</p> $\sqrt{\sum_{j=1}^n  x_j ^2}$
$\ A\ _1$	<p>The one norm of matrix <math>A</math>, defined as:</p> $\max \left\{ \sum_{i=1}^m  a_{ij} , 1 \leq j \leq n \right\}$

Item	Description
$\ A\ _2$	The spectral norm of matrix $A$ , defined as: $\max\{\ Ax\ _2 : \ x\ _2 = 1\}$
$\ A\ _F$	The Frobenius norm of matrix $A$ , defined as: $\sqrt{\sum_{i=1}^m \sum_{j=1}^n  a_{ij} ^2}$
$\ A\ _\infty$	The infinity norm of matrix $A$ , defined as: $\max \left\{ \sum_{j=1}^n  a_{ij} , 1 \leq i \leq m \right\}$
$A^{-1}$	The inverse of matrix $A$
$A^{-T}$	The transpose of $A$ inverse
$ A $	The determinant of matrix $A$
$m$ by $n$ matrix $A$	Matrix $A$ has $m$ rows and $n$ columns
$\sin a$	The sine of $a$
$\cos b$	The cosine of $b$
$\text{SIGN}(a)$	The sign of $a$ ; the result is either + or –
address $\{a\}$	The storage address of $a$
$\text{size}(a, \text{dim})$	The result equals the number of elements in $a$ along a specified dimension $\text{dim}$ or if $\text{dim}$ is not present the total number of array elements in $a$ .
$\text{max}(x)$	The maximum element in vector $x$
$\text{min}(x)$	The minimum element in vector $x$
$\text{ceiling}(x)$	The smallest integer that is greater than or equal to $x$
$\text{floor}(x)$	The largest integer that is not greater than $x$
$\text{iceil}(m,n)$	The smallest integer that is greater than or equal to $m/n$ ; that is, $\text{iceil}(m,n) = \text{ceiling}(m/n)$
$\text{ilcm}(i1,i2)$	The integer least common multiple of the integers, $i1$ and $i2$ .
$\text{int}(x), x > 0$	The largest integer that is less than or equal to $x$
$m \blacktriangleright (p, i)$	$m$ is mapped into $(p, i)$
$\text{mod}(x, m)$	$x$ modulo $m$ ; the remainder when $x$ is divided by $m$
$\infty$	Infinity
$\pi$	Pi, 3.14159265

## Interpreting the Subroutine Descriptions

This section explains how to interpret the information in the subroutine descriptions in Part 2 and 3 of this book. Each subroutine description explains the function(s) performed by the subroutine(s). It provides a data types table, showing how the data differs for each subroutine. It also contains sections that are described below.

### Syntax

This section shows the syntax for the Fortran, C, and C++ calling statements.

## Fortran, C, and C++ Syntax

This section shows the syntax for the Fortran, C, and C++ calling statements.

<b>Fortran</b>	CALL NAME-1   NAME-2   ...   NAME-n ( <i>arg-1, arg-2, ... , arg-m</i> )
<b>C and C++</b>	name-1   name-2   ...   name-n ( <i>arg-1, ... , arg-m</i> );

The syntax indicates:

- The programming language (Fortran, C, or C++)
- Each possible subroutine name that you can code in the calling sequence. Each name is separated by the | (or) symbol. You specify only one of these names in your calling sequence. (You do not code the | in the calling sequence.)
- The arguments, listed in the order in which you code them in the calling sequence. You must code them all in your calling sequence.

You can distinguish between input arguments and output arguments by looking at the “On Entry” and “On Return” sections, respectively. An argument used for both input and output is described in both the “On Entry” and “On Return” sections. In this case, the input value for the argument is overlaid with the output value.

## Fortran 90 Syntax

This shows the syntax for the Fortran 90 calling statements.

<b>Fortran 90</b>	<b>Equations or Cases</b>	CALL NAME ( <i>req-1, ... , req-m</i> )  CALL NAME ( <i>req-1, ... , req-m, opt-1, ... , opt-l</i> )
-------------------	-------------------------------	--

The syntax indicates:

- The programming language (Fortran 90)
- The Parallel ESSL subroutine name, which is a generic name for one or more functions.
- The arguments in the calling sequence.

The first calling sequence shows the arguments required when coding your program. The second calling sequence shows all the arguments, required and optional. The subroutine assigns a default value for any optional argument that is not present.

You can distinguish between input arguments and output arguments by looking at the “On Entry” and “On Return” sections, respectively. An argument used for both input and output is described in both the “On Entry” and “On Return” sections. In this case, the input value for the argument is overlaid with the output value.

## On Entry

This lists the input arguments, which are the arguments you pass to the subroutine. Each argument description first gives the meaning of the argument, and then gives the form of data required for the argument. (To help you avoid errors, output arguments are included, with a reference to the On Return section.)

## On Return

This lists the output arguments, which are the arguments passed back to your program from the subroutine. Each argument description first gives the meaning of the argument, and then gives the form of data passed back to your program for the argument.

## Notes and Coding Rules

The notes describe any programming considerations and restrictions that apply to the arguments or the data for the arguments. There may be references to other parts of the book for further information.

## Error Conditions

These are all the Parallel ESSL run-time errors that can occur in the subroutine. They are organized under the headings, “Computational Errors”, “Input Argument Errors”, “Resource Errors”, “Communications Errors”, and “Miscellaneous Errors”.

## Example

The two reference sections in this book contain different types of examples.

### Fortran Examples

The examples in Part 2 of this book show how you would call the subroutine in a Fortran program. Each example includes:

- A description of the salient features of the example
- The calling sequence, coded in Fortran
- The input and output data distributed across a process grid



---

## Summary of Changes

The following sections summarize changes to Parallel ESSL and the Parallel ESSL documentation for each new release or major service update for a given product version. Within each book in the library, a vertical line to the left of text and illustrations indicates technical changes or additions made to the previous edition of the book.

### **Summary of changes for Parallel ESSL for AIX, Version 3 Release 2 as updated, April 2005**

This release of Parallel ESSL for AIX includes the following changes:

- On the AIX 5.2 64-bit kernel, Parallel ESSL for AIX now runs on clusters that include:
  - POWER5 servers connected with an IBM @server pSeries® High Performance Switch (HPS)
  - BladeCenter JS20 servers and POWER5 servers connected with a Myrinet-2000 Switch with Myrinet/PCI-X adapters
- Parallel ESSL for AIX now supports AIX 5.3 on standalone clusters or clusters connected via a LAN-supporting IP.
- Parallel ESSL now provides the C interface for the BLACS.

For more details, see “Hardware and Software Products That Can Be Used with Parallel ESSL” on page 5.

### **Summary of changes for Parallel ESSL for Linux on POWER, Version 3 Release 2 as updated, December 2004**

This release of Parallel ESSL for Linux on POWER includes the following changes:

- Parallel ESSL for Linux™ on POWER now runs on the SuSE Linux Enterprise Server 9 for POWER (SLES9) operating system and supports clusters of IBM POWER4, POWER5, and BladeCenter JS20 servers connected with a Myrinet-2000 switch with Myrinet/PCI-X adapters.
- The C interface for the BLACS is now available.

### **Summary of changes for Parallel ESSL for AIX, Version 3 Release 1 as updated, October 2003**

This release of Parallel ESSL for AIX includes the following changes:

- Parallel ESSL for AIX supports the IBM @server pSeries High Performance Switch (HPS) and can coexist in clusters managed by CSM.
- Parallel ESSL for AIX supports AIX 5L Version 5.2 with the 5200-01 Recommended Maintenance package (program number 5765-E62). The 32-bit kernel is only supported on standalone clusters or clusters connected via a LAN supporting IP.
- Parallel ESSL for AIX includes all of the subroutines in Parallel ESSL for Linux on pSeries, Version 3 Release 1.

- The Parallel ESSL Serial Libraries are used with the Parallel Environment MPI Signal Handling Library. Parallel Environment 4.1 provides only binary compatibility support for the MPI Signal Handling Library. Existing applications that use the Parallel ESSL Serial Libraries will continue to run, but if you are creating new applications you should use the Parallel ESSL SMP Libraries.
- Parallel ESSL for AIX provides manpages of the subroutine descriptions.
- Parallel ESSL for AIX documentation is no longer being shipped with the product. It can be viewed or downloaded from the URLs listed in Table 6 on page 11.
- The Parallel ESSL for AIX software license agreement is now shipped, viewed, and accepted electronically. (See the *Parallel ESSL for AIX Installation Guide* for details.)

### **Summary of changes for Parallel ESSL for Linux on pSeries, Version 3 Release 1 as updated, September 2003**

This release of Parallel ESSL for Linux on pSeries includes the following changes:

- Parallel ESSL for Linux on pSeries runs on the SuSE Linux Enterprise Server 8 for pSeries (SLES8) operating system and supports the Myrinet-2000 switch with Myrinet/PCI-X adapters.
- Parallel ESSL for Linux on pSeries includes all of the subroutines in Parallel ESSL for AIX, Version 2 Release 3, plus all of the following:
  - New Dense Linear Algebraic Equation Subroutines:
    - PDPOTRI and PZPOTRI; see the subroutine description for Positive Definite Real Symmetric or Complex Hermitian Matrix Inverse on page 469.
    - PDPOCON and PZPOCON; see the subroutine description for Estimation of the Reciprocal of the Condition Number of a Positive Definite Real Symmetric or Complex Hermitian Matrix on page 476.
  - New Eigensystems Analysis Subroutines:
    - PZGEBRD; see the subroutine description for Reduce a General Matrix to Bidiagonal Form on page 762.
    - PDGESVD and PZGESVD; see the subroutine description for Singular Value Decomposition of a General Matrix on page 780.
  - New Utility subroutines:
    - DESCINIT; see the subroutine description for Initialize a Type-1 Array Descriptor with Error Checking on page 849.
    - DESCSET; see the subroutine description for Initialize a Type-1 Array Descriptor on page 852.
    - ICEIL; see the subroutine description for Compute the Ceiling of the Division of Two Integers on page 855.
    - ILCM; see the subroutine description for Compute the Least Common Multiple of Two Positive Integers on page 856.
    - INDYG2L; see the subroutine description for Compute the Local Row or Column Index of a Global Element of a Block-Cyclically Distributed Matrix on page 857.
    - INDYG2P; see the subroutine description for Compute the Process Row or Column Index of a Global Element of a Block-Cyclically Distributed Matrix on page 859.



- INDXL2G; see the subroutine description for Compute the Global Row or Column Index of a Local Element of a Block-Cyclically Distributed Matrix on page 861.
  - INFOG1L; see the subroutine description for Compute the Starting Local Row or Column Index and Process Row or Column Index of a Global Element of a Block-Cyclically Distributed Matrix on page 863.
  - INFOG2L; see the subroutine description for Compute the Starting Local Row and Column Indices and the Process Row and Column Indices of a Global Element of a Block-Cyclically Distributed Matrix on page 865.
  - PDLANSY, PZLANSY, and PZLANHE; see the subroutine description for Real Symmetric, Complex Symmetric, or Complex Hermitian Matrix Norm on page 879.
- Parallel ESSL for Linux on pSeries provides manpages of the subroutine descriptions.
  - Parallel ESSL for Linux on pSeries documentation is not shipped with the product. It can be viewed or downloaded from the URLs listed in Table 6 on page 11.
  - The Parallel ESSL for Linux on pSeries software license agreement is shipped, viewed, and accepted electronically. (See the *Parallel ESSL for Linux on pSeries Installation Guide* for details.)



---

## Part 1. Guide Information

This part of the book is organized into five chapters, providing guidance information on how to use Parallel ESSL. It is organized as follows:

- Overview, Requirements, and List of Subroutines
- Distributing Your Data
- Coding and Running Your Program
- Migrating Your Program
- Using Error Handling



---

## Chapter 1. Overview, Requirements, and List of Subroutines

This chapter introduces you to the IBM Parallel Engineering and Scientific Subroutine Library (Parallel ESSL) product.

---

### Overview of Parallel ESSL

Parallel ESSL is a scalable mathematical subroutine library that supports parallel processing applications on clusters of processor nodes optionally connected by a high-performance switch. Parallel ESSL supports the Single Program Multiple Data (SPMD) programming model using the Message Passing Interface (MPI) library.

Parallel ESSL provides subroutines in the following computational areas:

- Level 2 Parallel Basic Linear Algebra Subprograms (PBLAS)
- Level 3 PBLAS
- Linear Algebraic Equations
- Eigensystem Analysis and Singular Value Analysis
- Fourier Transforms
- Random Number Generation

For communication, Parallel ESSL includes the Basic Linear Algebra Communications Subprograms (BLACS), which use MPI. For computations, Parallel ESSL uses the ESSL subroutines.

The Parallel ESSL subroutines can be called from 32-bit- and 64-bit-environment application programs written in Fortran, C, and C++.

Parallel ESSL subroutines run under the AIX and Linux operating systems:

#### On AIX only

The **Parallel ESSL SMP Libraries** are provided for use with the Parallel Environment MPI threads library. You can run single or multithreaded US or IP applications on all types of nodes. However, you cannot simultaneously call Parallel ESSL from multiple threads. Use these Parallel ESSL libraries if you are using both PE MPI and LAPI. The SMP library is for use on the POWER™ and PowerPC® (for example, POWER4™) SMP processors.

#### On AIX and Linux

The **Parallel ESSL GM Libraries** are provided for use with the MPICH-GM library (see “Related Publications” on page 1002) and the Myrinet-2000 switch with Myrinet/PCI-X adapters. IP is not supported. You can run only single-threaded applications.

To order Parallel ESSL product, specify the appropriate program number as listed below:

**IBM Parallel ESSL for AIX**    5765-F84

**IBM Parallel ESSL for Linux**    5765-G18

## How Parallel ESSL Works

Parallel ESSL (which supports the SPMD programming model) uses MPI for communication during parallel processing and runs on clusters of processor nodes optionally connected by a high-performance switch.

A parallel program, such as yours with calls to the Parallel ESSL subroutines, executes as a number of individual, but related, **parallel tasks** on a number of your system's processor nodes. The group of parallel tasks is called a **partition**. The parallel tasks of your partition can communicate to exchange data or synchronize execution.

Your system may have an optional high-performance switch for communication. The switch increases the speed of communication between nodes. This helps your application program, as well as the Parallel ESSL subroutines, achieve maximum performance.

**Parallel ESSL assumes that the application program is using the SPMD programming model**, where the programs running the parallel tasks of your partition are identical. The tasks, however, work on different sets of data.

### Coding Your Program

The application developer creates a parallel program's source code, including calls to Parallel ESSL BLACS or MPI routines. These calls enable the parallel processes of your partition to communicate data and coordinate their execution.

Details on what other specific coding additions are required when using Parallel ESSL are given in Chapter 3, "Coding and Running Your Program," on page 77.

### Distributing Your Data

Your global data structures (vectors, matrices, or sequences) must be distributed across your processes prior to calling the Parallel ESSL subroutines.

Because data is distributed for both input and output, no implicit bottleneck is created by an initial scatter or ending gather operation. Parallel ESSL works in true SPMD mode, where each process operates only on a portion of the data. Also, the input and output data may be too large to collectively reside on a single node; therefore, problems associated with the storage limitations of a single processor node are eased by performing the computation in actual SPMD fashion.

See Chapter 2, "Distributing Your Data," on page 21 for details on distributing your data.

### Running and Testing

After writing the parallel application program containing calls to the Parallel ESSL subroutines, the developer then begins a cycle of modification and testing. The application program is run using the following products:

	<b>On AIX only</b>	Parallel Environment (PE)
	<b>On AIX and Linux</b>	MPICH-GM

These products include a number of **compiler scripts**, **environment variables**, and **command-line flags**, which may be used to set up your execution environment. (For example, before you execute a program, you need to set the size of your partition—the number of parallel tasks—by setting the appropriate environment variables or their command-line flags.)

For further details on PE and its various capabilities, see the PE manuals available at the URLs listed in Table 6 on page 11. For more information about MPI, see references [40] and [48]. For more information about MPICH-GM, see the URL listed in “Related Publications” on page 1002.

### **Tuning for Performance**

Once the parallel program is debugged, you now want to tune the program for optimal performance. This is an important step of the process, because performance is the key reason for using the Parallel ESSL subroutines. To tune and analyze programs with calls to the Parallel ESSL subroutines, you may wish to use the tools provided by PE. For details, see the PE manuals available at the URLs listed in Table 6 on page 11.

### **Accuracy of the Computations**

Parallel ESSL provides accuracy comparable to libraries using equivalent algorithms with identical precision formats. The data types operated on are ANSI/IEEE 64-bit binary floating-point format and 32-bit integer. See the *ANSI/IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985* for more detail.

### **The Fortran Language Interface to the Parallel ESSL Subroutines**

The Parallel ESSL subroutines follow standard Fortran calling conventions. When Parallel ESSL subroutines are called from a C or C++ program, the Fortran conventions must be used. This applies to all aspects of the interface, such as the linkage conventions and the data conventions. For example, array ordering must be consistent with Fortran array ordering techniques. Data and linkage conventions for each language are given in the *ESSL Guide and Reference*.

---

## **Hardware and Software Products That Can Be Used with Parallel ESSL**

This section describes the hardware and software products you can use with Parallel ESSL, as well as those products for installing Parallel ESSL and displaying the online documentation. It is divided into the following sections:

- “Hardware Products Supported by Parallel ESSL” on page 6
- “Operating Systems Supported by Parallel ESSL” on page 6
- “Software Products Required by Parallel ESSL” on page 7
- “Thread Safety and Parallel ESSL” on page 10
- “Installation and Customization of Parallel ESSL” on page 10

## Hardware Products Supported by Parallel ESSL

Parallel ESSL runs on the following hardware platforms:

*Table 1. Hardware platforms supported by Parallel ESSL*

Product	Supported Hardware
Parallel ESSL for AIX	<p>Clusters of:</p> <ul style="list-style-type: none"> <li>• POWER4 servers connected with a High Performance switch</li> <li>• POWER5 servers connected with a High Performance switch</li> <li>• POWER3 and POWER4 servers connected with an SP Switch2</li> <li>• BladeCenter JS20 processors and POWER5 servers connected with a Myrinet-2000 Switch with Myrinet/PCI-X adapters. (See Note 2.)</li> </ul>
Parallel ESSL for Linux	<p>Clusters of POWER4, POWER5, and BladeCenter JS20 servers connected with a Myrinet-2000 Switch with Myrinet/PCI-X adapters. (See Note 2.)</p>
<p><b>Notes:</b></p> <ol style="list-style-type: none"> <li>1. 64-bit applications require 64-bit hardware.</li> <li>2. IP is not supported.</li> </ol>	

## Operating Systems Supported by Parallel ESSL

Parallel ESSL runs in the following operating system environments:

*Table 2. Operating systems supported by Parallel ESSL*

Product	Supported Environment
Parallel Engineering and Scientific Subroutine Library for AIX, Version 3 Release 2 (program number 5765-G84)	AIX 5L™ Version 5.2 with the appropriate Recommended Maintenance Package, or later. (See Notes 1, 2, and 4.)
	AIX 5L Version 5.3, or later. (See Note 3.)
Parallel ESSL for Linux	SuSE Linux Enterprise Server 9 for POWER (SLES9)
<p><b>Notes:</b></p> <ol style="list-style-type: none"> <li>1. When running on an SP™ system (SP Switch 2), PSSP Version 3.5 (program number 5765-D51) is also required. However, when running on a standalone or cluster of pSeries or RS/6000® workstations and/or servers, PSSP is not required.</li> <li>2. The 32-bit kernel is only supported on standalone clusters or clusters connected via a LAN-supporting IP.</li> <li>3. AIX 5.3 is only supported on standalone clusters or clusters connected via a LAN-supporting IP; i.e., the High Performance Switch and the Myrinet-2000 switch are not supported.</li> <li>4. Myrinet-2000 switch support requires AIX 5L Version 5.2 with the 5200-04 Recommended Maintenance Package, or later. High Performance Switch support requires AIX 5L Version 5.2 with the 5200-05 Recommended Maintenance Package, or later.</li> </ol>	



## Software Products Required by Parallel ESSL

This section describes the software products that are required by Parallel ESSL. It is divided into the following sections:

- “Software Products Required by Parallel ESSL for AIX”
- “Software Products Required by Parallel ESSL for Linux” on page 8

### Software Products Required by Parallel ESSL for AIX

Parallel ESSL for AIX requires the software products shown in Table 3 for compiling and running.

ESSL for AIX must be ordered separately.

To assist C and C++ users, two header files are provided with the Parallel ESSL for AIX product. Use of these files is described in “Running Your Program on AIX” on page 91.

To assist Fortran 90 sparse linear algebraic equation users, module files are provided with the Parallel ESSL for AIX product. Use of this file is described in “Using Extrinsic Procedures—The Fortran 90 Sparse Linear Algebraic Equation Subroutines” on page 86.

**Required Software Products:** The following table lists the required software products for Parallel ESSL for AIX.

Table 3. Required Software Products for Parallel ESSL for AIX

Purpose	Required Software Product
For Compiling	IBM XL Fortran Enterprise Edition Version 9.1 for AIX (program number 5724-I08)
	or
	IBM XL C/C++ Enterprise Edition Version 7.0 for AIX (program number 5724-I11)
	or
	IBM XL C Enterprise Edition for AIX, Version 7.0 (program number 5724-I10)

Table 3. Required Software Products for Parallel ESSL for AIX (continued)

Purpose	Required Software Product
For Linking, Loading, or Running	<p>IBM XL Fortran Enterprise Edition Run-Time Environment Version 9.1 for AIX. (See Note 1.)</p> <p><b>and</b></p> <p>ESSL for AIX, Version 4.2 (program number 5765-F82). (See Note 2.)</p> <p><b>and</b></p> <p>C libraries. (See Note 3.)</p> <p><b>and</b></p> <p>One of the following:</p> <p><b>For IBM HPS, SP Switch 2, and IP mode</b> Parallel Environment for AIX, Version 4.2 (program number 5765-F83)</p> <p><b>For Myrinet-2000 switch</b> The following Myricom packages: GM 2.1 or later, which can be downloaded in binary (<b>installp</b>) format from: <a href="http://www.myri.com/scs">http://www.myri.com/scs</a></p> <p><b>and</b></p> <p>MPICH-GM 1.2.6..14-1 shared libraries. To learn how to obtain instructions for building these, see the Parallel ESSL “FAQs” section of the IBM @server Cluster Information Center at the URL listed in Table 6 on page 11.</p>
<p><b>Notes:</b></p> <ol style="list-style-type: none"> <li>1. IBM XL Fortran Enterprise Edition Run-Time Environment Version 9.1 for AIX is automatically shipped with the compiler. It is also available for downloading from the following Web site: <a href="http://www.ibm.com/software/awdtools/fortran">http://www.ibm.com/software/awdtools/fortran</a></li> <li>2. ESSL for AIX must be ordered separately.</li> <li>3. The AIX product includes the C and math libraries in the Application Development Toolkit.</li> </ol>	

## Software Products Required by Parallel ESSL for Linux

Parallel ESSL for Linux requires the software products shown in Table 4 on page 9 for compiling and running.

To assist C and C++ users, two header files are provided. Use of these files is described in “Running Your Program on Linux” on page 94.

To assist Fortran 90 sparse linear algebraic equation users, module files are provided with the Parallel ESSL product. Use of this file is described in “Using Extrinsic Procedures—The Fortran 90 Sparse Linear Algebraic Equation Subroutines” on page 86.

**Required Software Products:** The following table lists the required software products for Parallel ESSL for Linux on POWER:

Table 4. Required Software Products for Parallel ESSL for Linux on POWER

Purpose	Required Software Product
For Compiling	<p><b>For RHEL3 and SLES9</b>            IBM XL Fortran Advanced Edition Version 9.1 for Linux  <i>or</i>            IBM XL C/C++ Advanced Edition Version 7.0 for Linux</p> <p><b>For RHEL4</b>            IBM XL Fortran Advanced Edition Version 9.1.1 for Linux  <i>or</i>            IBM XL C/C++ Advanced Edition Version 7.0.1 for Linux</p>
For Linking, Loading, or Running	<p><b>For RHEL3 and SLES9</b>            IBM XL Fortran Advanced Edition Run-Time Environment Version 9.1 for Linux<sup>1</sup>  <b>and</b>            ESSL for Linux on POWER, Version 4 Release 2 (program number 5765-G17)<sup>2</sup>  <b>and</b>            GCC 3.3.3 32-bit libraries on SLES9            GCC 9.0.42 64-bit libraries on SLES9</p> <p><b>For RHEL4</b>            IBM XL Fortran Advanced Edition Run-Time Environment Version 9.1.1 for Linux<sup>1</sup>  <b>and</b>            ESSL for Linux on POWER, Version 4 Release 2 (program number 5765-G17)<sup>2</sup>  <b>and</b>  <b>and</b>            the following Myricom packages in source format, which can be downloaded from:  <a href="http://www.myri.com/scs">http://www.myri.com/scs</a>  <ul style="list-style-type: none"> <li>• GM 2.0.13 for Linux or later</li> <li>• MPICH-GM 1.2.6..13</li> </ul>           Selected levels of GM, MPICH, and MPICH-GM RPMs in binary format are available for downloading at:  <a href="http://ppclinux.ncsa.uiuc.edu">http://ppclinux.ncsa.uiuc.edu</a></p>
<p><b>Notes:</b></p> <ol style="list-style-type: none"> <li>1. IBM XL Fortran Advanced Edition Run-Time Environment for Linux is automatically shipped with the compiler. It is also available for downloading from the following Web site:  <a href="http://www.ibm.com/software/awdtools/fortran">http://www.ibm.com/software/awdtools/fortran</a></li> <li>2. ESSL for Linux must be ordered separately.</li> <li>3. Optional RPMs are required for building applications. For details, consult the Linux and compiler documentation.</li> </ol>	

## Thread Safety and Parallel ESSL

This section describes thread safety as it relates to Parallel ESSL. It is divided into the following sections:

- “Thread Safety and Parallel ESSL for AIX”
- “Thread Safety and Parallel ESSL for Linux”

### Thread Safety and Parallel ESSL for AIX

The Parallel ESSL SMP libraries are not thread safe; however, they are thread tolerant and can therefore be called from a single thread of a multithreaded application. Multiple simultaneous calls to the Parallel ESSL SMP libraries from different threads of a single process can cause unpredictable results.

The Parallel ESSL GM libraries are neither thread safe nor thread tolerant. Use them only when running single-threaded applications.

For more information on Thread Programming Concepts, see *IBM AIX General Programming Concepts: Writing and Debugging Programs*.

### Thread Safety and Parallel ESSL for Linux

Parallel ESSL for Linux is neither thread safe nor thread tolerant; therefore, you can run only single-threaded applications.

## Installation and Customization of Parallel ESSL

This section describes the installation and customization of Parallel ESSL. It is divided into the following sections:

- “Installation and Customization of Parallel ESSL for AIX”
- “Installation and Customization of Parallel ESSL for Linux”

### Installation and Customization of Parallel ESSL for AIX

Parallel ESSL is distributed on a compact disc (CD). The *Parallel ESSL for AIX Installation Guide* provides the detailed information you need to install Parallel ESSL on AIX.

The Parallel ESSL product is packaged in accordance with the AIX guidelines. The product can be installed using the **smit** command, and it can be installed on multiple nodes using the **dsh** command and the **installp** command.

### Installation and Customization of Parallel ESSL for Linux

Parallel ESSL for Linux is distributed on a CD. The *Parallel ESSL for Linux on pSeries Installation Guide* provides the detailed information you need to install Parallel ESSL on Linux.

The Parallel ESSL product is packaged as RPM packages. The product can be installed using the **rpm** command, as described at the following URL:

<http://www.rpm.org/>

---

## Software Products Required for Displaying Parallel ESSL Documentation

The software products needed to display Parallel ESSL online information are listed in Table 5.

Table 5. Software needed to display various formats of online information

Format of online information	Software needed
HTML	HTML document browser (such as Microsoft® Internet Explorer)
PDF	Adobe Acrobat Reader, which is freely available for downloading from the Adobe web site at: <a href="http://www.adobe.com">http://www.adobe.com</a>
Manpages	<p>No additional software needed. To display a specific manpage, use the <b>man</b> command as follows:</p> <ul style="list-style-type: none"><li>• <b>man</b> <i>subroutine-name</i></li></ul> <p><b>Note:</b> These manpages will be installed in the following directory:</p> <p><b>On AIX</b>                /usr/share/man/cat3</p> <p><b>On Linux</b>            /usr/share/man/man3</p> <p>In order for manpages to display properly on Linux, the LANG environment variable must be set to either of the following values: <b>C</b> or <b>en_US.iso885915</b>.</p> <p>The manpages provided by ScaLAPACK are installed in the /usr/share/man/man1 directory. By default, Parallel ESSL manpages will be displayed rather than PBLAS or ScaLAPACK manpages with the same names. If you want to access the PBLAS or ScaLAPACK manpages, you must set the MANPATH environment variable. See the documentation for the <b>man</b> command.</p>

---

## Parallel ESSL Internet Resources

Parallel ESSL documentation, as well as other related information, can be displayed or downloaded from the Internet at the URLs listed in Table 6.

Table 6. Online resources for information related to ESSL products

Web Site	Type of Information Provided	File Formats Available	
		PDF	HTML
IBM @server Cluster Information Center: <a href="http://publib.boulder.ibm.com/infocenter/clresctr/index.jsp">http://publib.boulder.ibm.com/infocenter/clresctr/index.jsp</a>	Documentation for IBM clustered-server and pSeries software products	Yes	Yes
IBM Publications Center: <a href="http://w3.ehone.ibm.com/public/applications/publications/cgibin/pbi.cgi">http://w3.ehone.ibm.com/public/applications/publications/cgibin/pbi.cgi</a>	Documentation for any IBM product	Yes	No

---

## Getting on the ESSL Mailing List

Late breaking information about ESSL products can be obtained by being placed on the ESSL mailing list. Users on the mailing list will receive information about new ESSL function and may receive customer satisfaction surveys and requirements surveys, to provide feedback to ESSL Development on the product and user requirements.

You can be placed on the mailing list by sending a request to either of the following, asking to be placed on the ESSL mailing list:

International Business Machines Corporation  
ESSL Development  
Department 85BA/Mail Station P963  
2455 South Rd.  
Poughkeepsie, NY 12601-5400  
e-mail: [essl@us.ibm.com](mailto:essl@us.ibm.com)

**Note:** To be withdrawn from the ESSL mailing list, send an e-mail to the address above.

When requesting to be placed on the mailing list or asking any questions, please provide the following information:

- Your name
- The name of your company
- Your mailing address
- Your Internet address
- Your phone number

---

## BLACS—Usage in Parallel ESSL for Communication

The Basic Linear Algebra Communication Subprograms (BLACS) provide ease-of-use and portability for message passing in parallel linear algebra programs. The BLACS efficiently support not only point-to-point operations between processes on a logical two-dimensional process grid, but also collective communications on such grids, or within just a grid row or column (a one-dimensional process grid).

Most communication packages, such as MPI, require an address and a length to be sent; therefore, they are classified as having operations based on vectors. In programming linear algebra problems, however, it is preferable to express all operations in terms of matrices. Vectors and scalars are simply subclasses of matrices. The BLACS operate on matrices, as defined by an address, column size, row size, leading dimension, and so forth.

Parallel ESSL includes the following interfaces for the BLACS:

- Fortran interface for the BLACS
- C interface for the BLACS

A BLACS quick reference guide can be found in Appendix A, “BLACS Quick Reference Guide,” on page 891.

An example of the usage of BLACS in a Fortran 90 program is shown in Appendix B, “Sample Programs,” on page 895.

The BLACS are documented in references [6], [33], and [34].

---

## List of Parallel ESSL Subroutines

This section provides an overview of the subroutines in each of the areas of Parallel ESSL.

### Level 2 PBLAS

The Level 2 PBLAS include a subset of the standard set of distributed memory parallel versions of the Level 2 BLAS.

**Note:** These subroutines were designed in accordance with the proposed Level 2 PBLAS standard. (See references [15], [16], and [18].) If these subroutines do not comply with the standard as approved, IBM will consider updating them to do so.

If IBM updates these subroutines, the update could require modifications of the calling application program.

*Table 7. List of Level 2 PBLAS*

Descriptive Name	Long-Precision Subprogram	Page
Matrix-Vector Product for a General Matrix or Its Transpose	PDGEMV PZGEMV	131
Matrix-Vector Product for a Real Symmetric or a Complex Hermitian Matrix	PDSYMV PZHENV	154
Rank-One Update of a General Matrix	PDGER PZGERC PZGERU	168
Rank-One Update of a Real Symmetric or a Complex Hermitian Matrix	PDSYR PZHER	186
Rank-Two Update of a Real Symmetric or a Complex Hermitian Matrix	PDSYR2 PZHER2	197
Matrix-Vector Product for a Triangular Matrix or Its Transpose	PDTRMV PZTRMV	212
Solution of Triangular System of Equations with a Single Right-Hand Sides	PDTRSV PZTRSV	288

### Level 3 PBLAS

The Level 3 PBLAS include a subset of the standard set of distributed memory parallel versions of the Level 3 BLAS.

**Note:** These subroutines were designed in accordance with the proposed Level 3 PBLAS standard. (See references [15], [16], and [18].) If these subroutines do not comply with the standard as approved, IBM will consider updating them to do so.

If IBM updates these subroutines, the update could require modifications of the calling application program.

Table 8. List of Level 3 PBLAS

Descriptive Name	Long-Precision Subprogram	Page
Matrix-Matrix Product for a General Matrix, Its Transpose, or Its Conjugate Transpose	PDGEMM PZGEMM	239
Matrix-Matrix Product Where One Matrix is Real or Complex Symmetric or Complex Hermitian	PDSYMM PZSYMM PZHEMM	256
Triangular Matrix-Matrix Product	PDTRMM PZTRMM	276
Solution of Triangular System of Equations with Multiple Right-Hand Sides	PDTRSM PZTRSM	288
Rank-K Update of a Real or Complex Symmetric or a Complex Hermitian Matrix	PDSYRK PZSYRK PZHERK	301
Rank-2K Update of a Real or Complex Symmetric or a Complex Hermitian Matrix	PDSYR2K PZSYR2K PZHER2K	316
Matrix Transpose for a General Matrix	PDTRAN PZTRANC PZTRANU	336

## Linear Algebraic Equations

These subroutines consist of dense, banded, and sparse subroutines, and include a subset of the ScaLAPACK subroutines.

**Note:** The dense and banded linear algebraic equations subroutines were designed in accordance with the proposed ScaLAPACK standard. See references [10], [17], [19], [28], and [29]. If these subroutines do not comply with the standard as approved, IBM will consider updating them to do so.

If IBM updates these subroutines, the update could require modifications of the calling application program.

### Dense Linear Algebraic Equations

The dense linear algebraic equation subroutines provide:

- Solutions to linear systems of equations for real and complex general matrices, and their transposes, and for positive definite real symmetric and complex Hermitian matrices.
- Least squares solutions to linear systems of equations for real and complex general matrices.
- Inverse of real and complex general matrices and of positive definite real symmetric and complex Hermitian matrices.
- Condition number of real and complex general matrices and of positive definite real symmetric and complex Hermitian matrices.

Table 9. List of Dense Linear Algebraic Equation Subroutines

Descriptive Name	Long-Precision Subroutine	Page
General Matrix Factorization and Solve	PDGESV PZGESV	355



Table 9. List of Dense Linear Algebraic Equation Subroutines (continued)

Descriptive Name	Long-Precision Subroutine	Page
General Matrix Factorization	PDGETRF PZGETRF	370
General Matrix Solve	PDGETRS PZGETRS	381
General Matrix Inverse	PDGETRI PZGETRI	393
Estimate the Reciprocal of the Condition Number of a General Matrix	PDGECON PZGECON	402
General Matrix QR Factorization	PDGEQRF PZGEQRF	411
General Matrix Least Squares Solution	PDGELS PZGELS	421
Positive Definite Real Symmetric or Complex Hermitian Matrix Factorization and Solve	PDPOSV PZPOSV	435
Positive Definite Real Symmetric or Complex Hermitian Matrix Factorization	PDPOTRF PZPOTRF	449
Positive Definite Real Symmetric or Complex Hermitian Matrix Solve	PDPOTRS PZPOTRS	458
Positive Definite Real Symmetric or Complex Hermitian Matrix Inverse	PDPOTRI PZPOTRI	469
Estimation of the Reciprocal of the Condition Number of a Positive Definite Real Symmetric or Complex Hermitian Matrix	PDPOCON PZPOCON	476

## Banded Linear Algebraic Equations

The banded linear algebraic equation subroutines provide solutions to linear systems of equations for real positive definite symmetric band matrices, real general tridiagonal matrices, diagonally-dominant real general tridiagonal matrices, and real positive definite symmetric tridiagonal matrices.

Table 10. List of Banded Linear Algebraic Equation Subroutines

Descriptive Name	Long- Precision Subroutine	Page
Positive Definite Symmetric Band Matrix Factorization and Solve	PDPBSV	486
Positive Definite Symmetric Band Matrix Factorization	PDPBTRF	498
Positive Definite Symmetric Band Matrix Solve	PDPBTRS	507
General Tridiagonal Matrix Factorization and Solve	PDGTSV	518
General Tridiagonal Matrix Factorization	PDGTTRF	532
General Tridiagonal Matrix Solve	PDGTTRS	548
Diagonally-Dominant General Tridiagonal Matrix Factorization and Solve	PDDTSV	518
Diagonally-Dominant General Tridiagonal Matrix Factorization	PDDTTRF	532

Table 10. List of Banded Linear Algebraic Equation Subroutines (continued)

Descriptive Name	Long- Precision Subroutine	Page
Diagonally-Dominant General Tridiagonal Matrix Solve	PDDTTRS	548
Positive Definite Symmetric Tridiagonal Matrix Factorization and Solve	PDPTSV	565
Positive Definite Symmetric Tridiagonal Matrix Factorization	PDPTTRF	579
Positive Definite Symmetric Tridiagonal Matrix Solve	PDPTTRS	591

### Fortran 90 Sparse Linear Algebraic Equation Subroutines

The Fortran 90 sparse linear algebraic equation subroutines provide solutions to linear systems of equations for a real general sparse matrix. The sparse utility subroutines provided in Parallel ESSL must be used in conjunction with the sparse linear algebraic equation subroutines.

Table 11. List of Fortran 90 Sparse Linear Algebraic Equation Subroutines

Descriptive Name	Long-Precision Subroutine	Page
Allocates Space for an Array Descriptor for a General Sparse Matrix	PADALL	607
Allocates Space for a General Sparse Matrix	PSPALL	609
Allocates Space for a Dense Vector	PGEALL	611
Inserts Local Data into a General Sparse Matrix	PSPINS	613
Inserts Local Data into a Dense Vector	PGEINS	617
Assembles a General Sparse Matrix	PSPASB	619
Assembles a Dense Vector	PGEASB	622
Preconditioner for a General Sparse Matrix	PSPGPR	624
Iterative Linear System Solver for a General Sparse Matrix	PSPGIS	627
Deallocates Space for a Dense Vector	PGEFREE	632
Deallocates Space for a General Sparse Matrix	PSPFREE	633
Deallocates Space for an Array Descriptor for a General Sparse Matrix	PADFREE	635

### Fortran 77 Sparse Linear Algebraic Equation Subroutines

The Fortran 77 sparse linear algebraic equation subroutines provide solutions to linear systems of equations for a real general sparse matrix. The sparse utility subroutines provided in Parallel ESSL must be used in conjunction with the sparse linear algebraic equation subroutines.

Table 12. List of The Fortran 77 Sparse Linear Algebraic Equation Subroutines

Descriptive Name	Long-Precision Subroutine	Page
Initializes an Array Descriptor for a General Sparse Matrix	PADINIT	643
Initializes a General Sparse Matrix	PDSPINIT	645
Inserts Local Data into a General Sparse Matrix	PDSPINS	647
Inserts Local Data into a Dense Vector	PDGEINS	652

Table 12. List of The Fortran 77 Sparse Linear Algebraic Equation Subroutines (continued)

Descriptive Name	Long-Precision Subroutine	Page
Assembles a General Sparse Matrix	PDSPASB	655
Assembles a Dense Vector	PDGEASB	659
Preconditioner for a General Sparse Matrix	PDSPGPR	661
Iterative Linear System Solver for a General Sparse Matrix	PDSPGIS	664

## Eigensystem Analysis and Singular Value Analysis

The eigensystems analysis subroutines provide solutions to the algebraic and generalized eigensystem analysis problem. The singular value analysis subroutines provide the singular value decomposition. These subroutines include a subset of the ScaLAPACK subroutines. See references [20] and [21].

**Note:** These subroutines were designed in accordance with the proposed ScaLAPACK standard. If these subroutines do not comply with the standard as approved, IBM will consider updating them to do so. If IBM updates these subroutines, the update could require modifications of the calling application program.

Table 13. List of Eigensystem Analysis and Singular Value Analysis Subroutines

Descriptive Name	Long-Precision Subroutine	Page
Selected Eigenvalues and, Optionally, the Eigenvectors of a Real Symmetric or Complex Hermitian Matrix	PDSYEVX PZHEEVX	677
Selected Eigenvalues and, Optionally, the Eigenvectors of a Real Symmetric or Complex Hermitian Positive Definite Generalized Eigenproblem	PDSYGVX PZHEGVX	698
Reduce a Real Symmetric or Complex Hermitian Matrix to Tridiagonal Form	PDSYTRD PZHETRD	724
Reduce a Real Symmetric or Complex Hermitian Positive Definite Generalized Eigenproblem to Standard Form	PDSYGST PZHEGST	739
Reduce a General Matrix to Upper Hessenberg Form	PDGEHRD	753
Reduce a General Matrix to Bidiagonal Form	PDGEBRD PZGEBRD	762
Singular Value Decomposition of a General Matrix	PDGESVD PZGESVD	780

## Fourier Transforms

The Fourier transform subroutines perform mixed-radix transforms in two and three dimensions. See references [1] and [3].

Table 14. List of Fourier Transform Subroutines

Descriptive Name	Short- Precision Subroutine	Long- Precision Subroutine	Page
Complex Fourier Transforms in Two Dimensions	PSCFT2	PDCFT2	800
Real-to-Complex Fourier Transforms in Two Dimensions	PSRCFT2	PDRCFT2	807

Table 14. List of Fourier Transform Subroutines (continued)

Descriptive Name	Short- Precision Subroutine	Long- Precision Subroutine	Page
Complex-to-Real Fourier Transforms in Two Dimensions	PSCRFT2	PDCRFT2	812
Complex Fourier Transforms in Three Dimensions	PSCFT3	PDCFT3	817
Real-to-Complex Fourier Transforms in Three Dimensions	PSRCFT3	PDRCFT3	825
Complex-to-Real Fourier Transforms in Three Dimensions	PSCRFT3	PDCRFT3	831

## Random Number Generation

The random number generation subroutine generates uniformly distributed random numbers.

Table 15. List of Random Number Generation Subroutines

Descriptive Name	Long-Precision Subroutine	Page
Uniform Random Number Generator	PDURNG	839

## Utilities

The utility subroutines perform general service functions that support Parallel ESSL.

Table 16. List of Utility Subroutines

Descriptive Name	Subprogram	Page
Determine the Level of Parallel ESSL Installed on Your System	IPESSL	847
Initialize a Type-1 Array Descriptor with Error Checking	DESCINIT	849
Initialize a Type-1 Array Descriptor	DESCSET	852
Compute the Ceiling of the Division of Two Integers	ICEIL	855
Compute the Least Common Multiple of Two Positive Integers	ILCM	856
Compute the Local Row or Column Index of a Global Element of a Block-Cyclically Distributed Matrix	INDXG2L	857
Compute the Process Row or Column Index of a Global Element of a Block-Cyclically Distributed Matrix	INDXG2P	859
Compute the Global Row or Column Index of a Local Element of a Block-Cyclically Distributed Matrix	INDXL2G	861
Compute the Starting Local Row or Column Index and Process Row or Column Index of a Global Element of a Block-Cyclically Distributed Matrix	INFOG1L	863
Compute the Starting Local Row and Column Indices and the Process Row and Column Indices of a Global Element of a Block-Cyclically Distributed Matrix	INFOG2L	865
Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process	NUMROC	868
General Matrix Norm	PDLANGE PZLANGE	872

Table 16. List of Utility Subroutines (continued)

Descriptive Name	Subprogram	Page
Real Symmetric, Complex Symmetric, or Complex Hermitian Matrix Norm	PDLANSY PZLANSY PZLANHE	879



---

## Chapter 2. Distributing Your Data

This chapter provides information on how to distribute data for your programs. The sections include:

- “Concepts”
- “Specifying and Distributing Data in Your Program” on page 26

---

### Concepts

This section describes the general concepts used in distributing data.

#### About Global Data Structures

Because the Parallel ESSL subroutines support the SPMD programming model, your global data structures (vectors, matrices, or sequences) must be distributed across your processes prior to calling the Parallel ESSL subroutines.

Conceptually, global data structures have a defined storage mode consistent with those used by the serial ESSL library, **except for symmetric tridiagonal matrices**. For Parallel ESSL, you must store symmetric tridiagonal matrices as described in this chapter in “Block-Cyclically Distributing a Symmetric Tridiagonal Matrix” on page 48. For how to store all other data structures when using Parallel ESSL, you should see the appropriate section in the *ESSL Guide and Reference*. The FFT-packed storage mode is a new storage mode for Parallel ESSL and is described in “Specifying Sequences for the Fourier Transforms” on page 64.

Global data structures must be mapped to local (distributed memory) data structures, according to the data distribution technique supported by the Parallel ESSL subroutines that you are using. These local data structures are called local arrays.

These data distribution techniques are described throughout this chapter and apply equally to real and complex data structures.

#### About Process Grids

A parallel machine with  $k$  processes is often thought of as a one-dimensional linear array of processes labeled 0, 1, ...,  $k-1$ . For performance reasons, it is sometimes useful to map this one-dimensional array into a logical two-dimensional rectangular grid, which is also referred to as process grid, of processes. The process grid can have  $p$  process rows and  $q$  process columns, where  $p \times q = k$ . A process can now be indexed by row and column,  $(i,j)$ , where  $0 \leq i < p$  and  $0 \leq j < q$ .

Table 17 shows six processes mapped into a process grid using row-major order. For message passing subroutines, the BLACS\_GRIDINIT default to map processes is row-major order. In this example, process  $t_3$  is mapped to  $P_{10}$ .

Table 17. Six Processes Mapped to a  $2 \times 3$  Process Grid Using Row-Major Order

$p,q$	0	1	2
0	$t_0$	$t_1$	$t_2$
1	$t_3$	$t_4$	$t_5$

Table 18 shows six processes mapped into a process grid using column-major order. In this example, process  $t_3$  is mapped to  $P_{11}$ .

*Table 18. Six Processes Mapped to a  $2 \times 3$  Process Grid Using Column-Major Order*

$p,q$	0	1	2
0	$t_0$	$t_2$	$t_4$
1	$t_1$	$t_3$	$t_5$

All the subroutines, except the Banded Linear Algebraic Equations and Fourier transform subroutines, can view the processes as a logical one- or two-dimensional process grid. The Banded Linear Algebraic Equations support one-dimensional process grids. The Fourier transform subroutines support one-dimensional, row-oriented process grids.

Each process has local memory, and all the processes are connected by a communication network (for example, a switch or Ethernet). In most cases  $k$  is less than or equal to the number of processor nodes that your job is running on. In special cases, however, the number of processes can be greater than the number of processor nodes.

## What to Do in Your Program

Prior to calling any of the subroutines, you must define your process grid and distribute your data according to the distribution technique required by the Parallel ESSL subroutine you are using.

The size and shape of the process grid and the way global data structures are distributed over the processes has a major impact on performance and scalability. For details, see “Coding Tips for Optimizing Parallel Performance” on page 77. Block-cyclic data distribution generally provides good load balancing for many linear algebra computations. All subroutines support block-cyclic data distributions, except the Fourier Transforms. These subroutines support only block distribution, which is a special case of block-cyclic data distribution.

Some of the data distribution techniques described in this chapter are illustrated in Appendix B, “Sample Programs,” on page 895.

## Block, Cyclic, and Block-Cyclic Data Distributions

In this section, three types of data distribution are described in algorithmic terms: block, cyclic, and block-cyclic. How these data distribution methods are used by Parallel ESSL is explained later in this chapter.

The example notation means the following:

- **B** represents the global block row numbers.
- **D** represents the global block column numbers.
- **p** represents the process row index.
- **q** represents the process column index.

### Distribution Techniques

An important aspect of the data distributions described here is that independent distributions are applied over each dimension of the data structure. The algorithms presented here for the vector in one dimension can, therefore, be used for the rows and columns of a matrix, or even for data structures with more dimensions.



Consider the distribution of a vector  $x$  of  $M$  data objects (elements) over  $P$  processes. This can be described by a mapping of the global index  $m$  ( $0 \leq m < M$ ) of a data object to an index pair  $(p, i)$ , where  $p$  ( $0 \leq p < P$ ) specifies the process to which the data object is mapped, and  $i$  specifies its location in the local array.

Two common distributions are the **block** and **cyclic**. The block distribution is often used when the computational load is distributed homogeneously over a regular data structure, such as a Cartesian grid. It assigns blocks of size  $r$  of the global vector to the processes. For block distribution, the mapping  $m \mapsto (p, i)$  is defined as:

- $m \mapsto (\text{floor}(m/L), m \bmod L)$

where  $L = \text{ceiling}(M/P)$ . The cyclic distribution (also known as the wrapped or scattered decomposition) is commonly used to improve load balance when the computational load is distributed inhomogeneously over a regular data structure. The cyclic distribution assigns consecutive entries of the global vector to successive processes. For cyclic distribution, the mapping  $m \mapsto (p, i)$  is defined as:

- $m \mapsto (m \bmod P, \text{floor}(m/P))$

Examples of block and cyclic distribution are shown in Figure 1 and Figure 2, where  $M = 23$  data objects are distributed over  $P = 3$  processes, using  $r = 8$  block size. As shown in the examples, there can be uneven distribution, where the last block is smaller than the others. A global block number  $B$  is shown for block distribution. For cyclic distribution, there is no concept of block numbers.

$m$	0	1	2	3	4	5	6	7		8	9	10	11	12	13	14	15		16	17	18	19	20	21	22
$p$	0	0	0	0	0	0	0	0		1	1	1	1	1	1	1	1		2	2	2	2	2	2	2
$i$	0	1	2	3	4	5	6	7		0	1	2	3	4	5	6	7		0	1	2	3	4	5	6
$B$	0	0	0	0	0	0	0	0		1	1	1	1	1	1	1	1		2	2	2	2	2	2	2

Figure 1. Block Distribution

$m$	0	1	2		3	4	5		6	7	8		9	10	11		12	13	14		15	16	17		18	19	20		21	22
$p$	0	1	2		0	1	2		0	1	2		0	1	2		0	1	2		0	1	2		0	1	2		0	1
$i$	0	0	0		1	1	1		2	2	2		3	3	3		4	4	4		5	5	5		6	6	6		7	7

Figure 2. Cyclic Distribution

The block-cyclic distribution is a generalization of the block and cyclic distributions, in which blocks of  $r$  consecutive data objects are distributed cyclically over the  $p$  processes. This can be described by a mapping of the global index  $m$  ( $0 \leq m < M$ ) of a data object to an index triplet  $(p, b, i)$ , where  $p$  ( $0 \leq p < P$ ) specifies the process to which the data object is mapped,  $b$  is the block number in process  $p$ , and  $i$  is the location in the block. For block-cyclic distribution, the mapping  $m \mapsto (p, b, i)$  is defined as:

- $m \mapsto (\text{floor}((m \bmod T)/r), \text{floor}(m/T), m \bmod r)$

where  $T = rP$ . (It should be noted that this reverts to the cyclic distribution when  $r = 1$  and a block distribution when  $r = L$ .) The inverse mapping to a global index  $(p, b, i) \mapsto m$  is defined by:

- $(p, b, i) \mapsto Br + i = pr + bT + i$

where  $B = p + bP$  is the global block number. An example of block-cyclic distribution is shown in Figure 3, where  $M = 23$  data objects are distributed over  $P = 3$  processes, using  $r = 2$  block size. As shown in the example, there can be uneven distribution, where the last block is smaller than the others. The inverse mapping is shown in the second part of the example. (This shows what is stored in the local array on each of the three processes.)

$m$	0 1 2 3 4 5	6 7 8 9 10 11	12 13 14 15 16 17	18 19 20 21 22
$p$	0 0 1 1 2 2	0 0 1 1 2 2	0 0 1 1 2 2	0 0 1 1 2
$b$	0 0 0 0 0 0	1 1 1 1 1 1	2 2 2 2 2 2	3 3 3 3 3
$i$	0 1 0 1 0 1	0 1 0 1 0 1	0 1 0 1 0 1	0 1 0 1 0
$B$	0 0 1 1 2 2	3 3 4 4 5 5	6 6 7 7 8 8	9 9 10 10 11

Figure 3. Block-Cyclic Distribution

$m$	0 1 6 7 12 13 18 19	2 3 8 9 14 15 20 21	4 5 10 11 16 17 22
$p$	0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1	2 2 2 2 2 2 2 2
$b$	0 0 1 1 2 2 3 3	0 0 1 1 2 2 3 3	0 0 1 1 2 2 3
$i$	0 1 0 1 0 1 0 1	0 1 0 1 0 1 0 1	0 1 0 1 0 1 0
$B$	0 0 3 3 6 6 9 9	1 1 4 4 7 7 10 10	2 2 5 5 8 8 11

Figure 4. Inverse Mapping of Block-Cyclic Distribution

In decomposing an  $m \times n$  matrix,  $A$ , independent block-cyclic distributions are applied in the row and column directions. Thus, suppose the matrix rows are distributed with block size  $r$  over  $P$  processes by the  $\lambda_{r,P}$  block-cyclic mapping, and the matrix columns are distributed with block size  $s$  over  $Q$  processes by the  $\psi_{s,Q}$  block-cyclic mapping. Then the matrix element indexed globally by  $(m, n)$  is mapped as follows:

$$m \mapsto^{\lambda} (p, b, i)$$

$$n \mapsto^{\psi} (q, d, j)$$

The distribution of the matrix can be regarded as the tensor product of the row and column distributions, which can be expressed as:

$$\bullet (m, n) \mapsto ((p, q), (b, d), (i, j))$$

The block-cyclic matrix distribution expressed above distributes blocks of size  $r \times s$  to a grid of  $P \times Q$  processes.

An example of block-cyclic distribution of an  $m \times n = 16 \times 30$  matrix with block size  $r \times s = 3 \times 4$  and a  $P \times Q = 2 \times 3$  process grid is shown in Figure 5 on page 25 and Figure 6 on page 25. The numbers in the leftmost column and on the top of the matrix represent the global row and column numbers **B** and **D**, respectively. Figure 5 on page 25 shows the assignment of global blocks (**B**,**D**) to processes (**P**,**Q**). Figure 6 on page 25 shows which global blocks each process contains.

In this example, the global matrix dimensions are not divisible by the respective block sizes. All the row blocks are of size 3, except the last row block, which only contains 1 row. All column blocks are of size 4, except the last column block, which contains 2 columns. For example, global block (5,0) is  $1 \times 4$ , global block (1,7) is  $3 \times 2$ , and global block (0,0) is  $3 \times 4$ . The global block (5,7) is  $1 \times 2$ . The asterisk (\*) in Figure 5 denotes which global blocks contain left over data; that is, the blocks that are not  $3 \times 4$ .

B,D	0	1	2	3	4	5	6	7
0	P <sub>00</sub>	P <sub>01</sub>	P <sub>02</sub>	P <sub>00</sub>	P <sub>01</sub>	P <sub>02</sub>	P <sub>00</sub>	P <sub>01</sub> *
1	P <sub>10</sub>	P <sub>11</sub>	P <sub>12</sub>	P <sub>10</sub>	P <sub>11</sub>	P <sub>12</sub>	P <sub>10</sub>	P <sub>11</sub> *
2	P <sub>00</sub>	P <sub>01</sub>	P <sub>02</sub>	P <sub>00</sub>	P <sub>01</sub>	P <sub>02</sub>	P <sub>00</sub>	P <sub>01</sub> *
3	P <sub>10</sub>	P <sub>11</sub>	P <sub>12</sub>	P <sub>10</sub>	P <sub>11</sub>	P <sub>12</sub>	P <sub>10</sub>	P <sub>11</sub> *
4	P <sub>00</sub>	P <sub>01</sub>	P <sub>02</sub>	P <sub>00</sub>	P <sub>01</sub>	P <sub>02</sub>	P <sub>00</sub>	P <sub>01</sub> *
5	P <sub>10</sub> *	P <sub>11</sub> *	P <sub>12</sub> *	P <sub>10</sub> *	P <sub>11</sub> *	P <sub>12</sub> *	P <sub>10</sub> *	P <sub>11</sub> *

Figure 5. Block Distribution Over a 2 by 3 Process Grid

B,D	0	3	6	1	4	7	2	5
0	P <sub>00</sub>			P <sub>01</sub>			P <sub>02</sub>	
2								
4								
1	P <sub>10</sub>			P <sub>11</sub>			P <sub>12</sub>	
3								
5								
	*	*	*	*	*	*	*	*

Figure 6. Data Distribution from a Process Point-of-View

B,D	0	3	6	1	4	7	2	5
0	a <sub>0:2,0:3</sub>	a <sub>0:2,12:15</sub>	a <sub>0:2,24:27</sub>	a <sub>0:2,4:7</sub>	a <sub>0:2,16:19</sub>	a <sub>0:2,28:29</sub> *	a <sub>0:2,8:11</sub>	a <sub>0:2,20:23</sub>
2	a <sub>6:8,0:3</sub>	a <sub>6:8,12:15</sub>	a <sub>6:8,24:27</sub>	a <sub>6:8,4:7</sub>	a <sub>6:8,16:19</sub>	a <sub>6:8,28:29</sub> *	a <sub>6:8,8:11</sub>	a <sub>6:8,20:23</sub>
4	a <sub>12:14,0:3</sub>	a <sub>12:14,12:15</sub>	a <sub>12:14,24:27</sub>	a <sub>12:14,4:7</sub>	a <sub>12:14,16:19</sub>	a <sub>12:14,28:29</sub> *	a <sub>12:14,8:11</sub>	a <sub>12:14,20:23</sub>
1	a <sub>3:5,0:3</sub>	a <sub>3:5,12:15</sub>	a <sub>3:5,24:27</sub>	a <sub>3:5,4:7</sub>	a <sub>3:5,16:19</sub>	a <sub>3:5,28:29</sub> *	a <sub>3:5,8:11</sub>	a <sub>3:5,20:23</sub>
3	a <sub>9:11,0:3</sub>	a <sub>9:11,12:15</sub>	a <sub>9:11,24:27</sub>	a <sub>9:11,4:7</sub>	a <sub>9:11,16:19</sub>	a <sub>9:11,28:29</sub> *	a <sub>9:11,8:11</sub>	a <sub>9:11,20:23</sub>
5	a <sub>15,0:3</sub> *	a <sub>15,12:15</sub> *	a <sub>15,24:27</sub> *	a <sub>15,4:7</sub> *	a <sub>15,16:19</sub> *	a <sub>15,28:29</sub> *	a <sub>15,8:11</sub> *	a <sub>15,20:23</sub> *

Figure 7. Distributed Matrix Elements from a Process Point-of-View

## Special Usage

The block-cyclic distribution can reproduce most of the data distributions commonly used in linear algebra computations on parallel computers. Some examples are:

- Block distribution in the row direction is obtained by  $Q = 1$  and  $r = \text{ceiling}(M/P)$ .
- Block distribution in the column direction is obtained by  $P = 1$  and  $s = \text{ceiling}(N/Q)$ .

- Block-cyclic distribution in the row direction is obtained by  $Q = 1$  and  $r < \text{ceiling}(M/P)$ . (You might use this for distributing a single block column to pass to Parallel ESSL.)
- Block-cyclic distribution in the column direction is obtained by  $P = 1$  and  $s < \text{ceiling}(N/Q)$ . (You might use this for distributing a single block row to pass to Parallel ESSL.)
- To achieve **fine** granularity of distribution in the following directions, specify:
  - For the row direction,  $r = 1$
  - For the column direction,  $s = 1$
  - In both directions,  $r = 1$  and  $s = 1$
- To achieve **coarse** granularity of distribution in the following directions, specify:
  - For the row direction,  $r = \text{ceiling}(M/P)$ .
  - For the column direction,  $s = \text{ceiling}(N/Q)$ .
  - In both directions,  $r = \text{ceiling}(M/P)$  and  $s = \text{ceiling}(N/Q)$ .

This section provided a detailed description of the distribution of vectors—one-dimensional data structures. Those same techniques were then applied to matrices—two-dimensional data structures—in the row and column directions. If you have data structures with three or more dimensions, you can use these same techniques by applying them in the direction of each dimension. For example, the block distribution of a three-dimensional sequence is described in “Three-Dimensional Sequences” on page 68.

---

## Specifying and Distributing Data in Your Program

This section describe the calling sequence arguments for vectors and matrices, and shows how to distribute vectors, matrices and sequences in your program for the following areas:

- For the Level 2 and 3 PBLAS, Dense Linear Algebraic Equations, and Eigensystem Analysis and Singular Value Analysis subroutines, see “Specifying Block-Cyclically-Distributed Vectors and Matrices.”
- For the Banded Linear Algebraic Equations, see “Specifying Block-Cyclically-Distributed Matrices for the Banded Linear Algebraic Equations” on page 29.
- For the Sparse Linear Algebraic Equations, see “Specifying Sparse Matrices for the Fortran 90 and Fortran 77 Sparse Linear Algebraic Equations” on page 58.
- For the Fourier Transforms, see “Specifying Sequences for the Fourier Transforms” on page 64.

An example of block-cyclic distribution of a global matrix in a Fortran 90 program in a message passing environment is shown in Appendix B, “Sample Programs.” See the following:

- The subroutine `get_diffusion_matrix` in “Module Fourier” on page 911, which shows how a local array can be assigned values.
- The subroutine `rlocal_to_rglobal` in “Module Scale” on page 919, which shows gathering the local portions of the block-cyclically-distributed real array to generate the corresponding global matrix.

## Specifying Block-Cyclically-Distributed Vectors and Matrices

For the Level 2 and 3 PBLAS, Dense Linear Algebraic Equations, and Eigensystem Analysis and Singular Value Analysis subroutines, certain calling sequence arguments are used to specify block-cyclically-distributed vectors or matrices.

## Calling Sequence Arguments for Block-Cyclically-Distributed Vectors and Matrices

Table 19 describes the arguments associated with a vector  $X$ . Table 20 describes the arguments associated with a matrix  $A$ .

Table 19. Calling Sequence Arguments for a Block-Cyclically-Distributed Vector

Argument	Meaning
$x$	is the local part of the global matrix $X$ . To determine the size of the local array for $X$ , see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28.
$ix$	is the row index of global matrix $X$ .
$jx$	is the column index of global matrix $X$ .
$desc\_x$	is the array descriptor for global matrix $X$ . (See Table 21.)
$incx$	Stride for global vector $X$ .

**Note:** A global vector of length  $n$  is distributed across process rows the same way as an  $n \times 1$  matrix is (in this case  $M\_X$  is  $n$  and  $N\_X$  is 1). A global vector of length  $n$  is distributed across process columns the same way as a  $1 \times n$  matrix is (in this case  $M\_X$  is 1 and  $N\_X$  is  $n$ ).

Table 20. Calling Sequence Arguments for a Block-Cyclically-Distributed Matrix

Argument	Meaning
$a$	is the local part of the global matrix $A$ . To determine the size of the local array for $A$ , see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28.
$ia$	is the row index of the global matrix $A$ .
$ja$	is the column index of the global matrix $A$ .
$desc\_a$	is the array descriptor for global matrix $A$ . (See Table 21.)

## Array Descriptors for Block-Cyclically-Distributed Matrices

An array descriptor, which is an integer array, is needed for each block-cyclically-distributed vector or matrix. The process grid definition and array descriptor are used to establish the mapping between the global vector or matrix and its corresponding process and distributed memory location.

Throughout this book, the  $\_$  (underscore) symbol in the array descriptor is followed by an  $X$  to indicate a vector or an  $A$  to indicate a matrix.

An example of setting up descriptor arrays in a Fortran 90 program is shown in Appendix B, “Sample Programs.” See the subroutines `initialize_rarray` and `initialize_carray` in “Module Scale” on page 919.

Table 21 shows the type-1 array descriptor, as it is used in the Level 2 and 3 PBLAS, Dense Linear Algebraic Equations, and Eigensystem Analysis and Singular Value Analysis subroutines.

Table 21. Type-1 Array Descriptor for Block-Cyclically Distributed Vector or Matrix

DESC_( )	Symbolic name	Meaning
1	DTYPE_	Descriptor type, where $DTYPE_=1$
2	CTXT_	BLACS context in which the global matrix is defined. (See “Using the BLACS” on page 80.)

Table 21. Type-1 Array Descriptor for Block-Cyclically Distributed Vector or Matrix (continued)

DESC_( )	Symbolic name	Meaning
3	M_	Number of rows in the global matrix
4	N_	Number of columns in the global matrix
5	MB_	Row block size
6	NB_	Column block size
7	RSRC_	The process row of the $p \times q$ process grid over which the first row of the global matrix is distributed
8	CSRC_	The process column of the $p \times q$ process grid over which the first column of the global matrix is distributed
9	LLD_	Leading dimension of the local array. (See “Determining the Number of Rows and Columns in Your Local Arrays.”) This value may be different on each process.

## Specifying Submatrices

After a global vector or matrix is block-cyclically distributed over a process grid, you may decide to use only a portion of the global data structure. This is called a submatrix. For examples of how to specify the calling sequence arguments, listed in Table 19 and Table 20, for a submatrix, see:

- “Example 1” on page 141
- “Example 2” on page 144
- “Example 1” on page 175
- “Example 1” on page 230
- “Example 1” on page 387
- “Example 1” on page 464

Suppose you decide to distribute your global vector or matrix over the process grid, starting at a process other than 0,0. For examples of how to set the array descriptor values, listed in Table 21, see:

- “Example 1” on page 374
- “Example 1” on page 387
- “Example 1” on page 464

## Determining the Number of Rows and Columns in Your Local Arrays

In a Parallel ESSL calling sequence, you specify an array that contains the local part of the global vector or matrix. To determine  $\text{LOCp}(M_)$  or  $\text{LOCq}(N_)$ , which are used in the subroutines descriptions in Part 2 of this book, you must make a call to NUMROC:

- For  $\text{LOCp}(M_)$ , which represents the number of rows that a process would receive if  $M_$  was distributed block-cyclically over the  $p$  rows of its process column, you specify:

$\text{LOCp}(M_) = \text{NUMROC}(M_, MB_, \text{myrow}, \text{RSRC_}, p)$

where:

- $M_$  is the number of rows in the global matrix.
- $MB_$  is the row block size.
- $\text{myrow}$  is the process row index. See “Using the BLACS” on page 80.

- RSRC\_ is the process row over which the first row of the global matrix is distributed.
- $p$  is the number of rows in the  $p \times q$  process grid.
- For LOCq(N\_), which represents the number of columns that a process would receive if N\_ was distributed block-cyclically over the  $q$  columns of its process row, you specify:
 

LOCq(N\_) = NUMROC (N\_, NB\_, mycol, CSRC\_, q)

 where:
  - N\_ is the number of columns in the global matrix.
  - NB\_ is the column block size.
  - mycol is the process column index. See “Using the BLACS” on page 80.
  - CSRC\_ is the process column over which the first column of the global matrix is distributed.
  - $q$  is the number of columns in the  $p \times q$  process grid.

## Specifying Block-Cyclically-Distributed Matrices for the Banded Linear Algebraic Equations

For the Banded Linear Algebraic Equations, certain calling sequence arguments are used to specify block-cyclically distributed matrices on one-dimensional process grids.

Although the global array is block-cyclically distributed, the actual submatrix used in computation is either block-row or block-column distributed. See the appropriate subroutine for restrictions.

### Symmetric Band Matrix

A symmetric band matrix must be distributed over a one-dimensional process grid:

- On a  $1 \times p$  process grid, the symmetric band matrix is block-cyclically distributed. In this case, either type-501 or type-1 array descriptor may be specified.
- On a  $p \times 1$  process grid, the symmetric band matrix is block-cyclically distributed as if the process grid is  $1 \times p$ . In this case, the type-501 array descriptor must be specified.

Table 22 describes the calling sequence arguments associated with a symmetric band matrix.

*Table 22. Calling Sequence Arguments for a Distributed Symmetric Band Matrix*

Argument	Meaning
$n$	is the order of the global symmetric band submatrix $A$ .
$a$	is the local part of the global symmetric band matrix $A$ .
$ja$	is the column index of the global symmetric band matrix $A$ .
$desc\_a$	is the array descriptor for the global symmetric band matrix $A$ . For more details, see Table 26 on page 31 and Table 21 on page 27.

### General Tridiagonal Matrix

A general tridiagonal matrix, represented as three vectors, must be distributed over a one-dimensional process grid using a block-cyclic data distribution. Because vectors are one-dimensional data structures, you can use type-501, type-502, or type-1 array descriptor regardless of whether the process grid is  $p \times 1$  or  $1 \times p$ . Table 23 on page 30 describes the calling sequence arguments associated with a



general tridiagonal matrix.

*Table 23. Calling Sequence Arguments for General Tridiagonal Matrix*

Argument	Meaning
$n$	is the order of the global general tridiagonal submatrix $A$ .
$dl, d, du$	is the local part of the global vectors. (The general tridiagonal matrix $A$ is stored in tridiagonal storage mode in $dl$ , $d$ , and $du$ .)
$ia$	is the row index of the global general tridiagonal matrix $A$ .
$desc\_a$	is the array descriptor for the global general tridiagonal matrix $A$ . For more details, see Table 26 on page 31, Table 21 on page 27, or Table 27 on page 32.

## Symmetric Tridiagonal Matrix

A symmetric tridiagonal matrix, represented as two vectors, must be distributed over a one-dimensional process grid using block-cyclic data distribution.

**Note:** For both serial ESSL and Parallel ESSL, the  $n-1$  elements of the equal off-diagonals of a symmetric tridiagonal matrix are stored in a one-dimensional vector of length  $n$ . To be compatible with ScaLAPACK, in Parallel ESSL, the off-diagonal is chosen to be the superdiagonal and is stored in elements  $ia$  through  $ia+n-2$ . In the serial ESSL library, the off-diagonal is chosen to be the subdiagonal and is stored in elements 2 through  $n$ .

Because vectors are one-dimensional data structures, you can use a type-501, type-502, or type-1 array descriptor regardless of whether the process grid is  $p \times 1$  or  $1 \times p$ . Table 24 describes the calling sequence arguments associated with a symmetric tridiagonal matrix.

*Table 24. Calling Sequence Arguments for a Symmetric Tridiagonal Matrix*

Argument	Meaning
$n$	is the order of the global symmetric tridiagonal submatrix $A$ .
$d, e$	is the local part of the global vectors. (The symmetric tridiagonal matrix $A$ is stored in parallel-symmetric-tridiagonal storage mode in $d$ and $e$ .)
$ia$	is the row index of the global symmetric tridiagonal matrix $A$ .
$desc\_a$	is the array descriptor for the global symmetric tridiagonal matrix $A$ . For more details, see Table 26 on page 31, Table 21 on page 27, or Table 27 on page 32.

## General Matrix Consisting of Multiple Right-Hand Sides

For the Banded Linear Algebraic Equations subroutines, a general matrix consisting of multiple right-hand sides must be distributed over a one-dimensional process grid:

- On a  $p \times 1$  process grid, the multiple right-hand sides is block-cyclically distributed. In this case either type-502 or type-1 array descriptor may be specified.
- On a  $1 \times p$  process grid, the multiple right-hand sides is block-cyclically distributed as if the process grid is  $p \times 1$ . In this case type-502 array descriptor must be specified.

Table 25 on page 31 describes the calling sequence arguments associated with the general matrix.



Table 25. Calling Sequence Arguments for a Matrix Containing the Multiple Right-Hand Sides

Argument	Meaning
$n$	is the number of rows in the global general submatrix $B$ .
$b$	is the local part of the global general matrix $B$ .
$ib$	is the row index of the global general matrix $B$ .
$desc\_b$	is the array descriptor for the global general matrix $B$ . For more details, see Table 27 on page 32 and Table 21 on page 27.

## Array Descriptors for Banded Matrices

An array descriptor, which is an integer array, is needed for each block-distributed matrix. The process grid definition and the array descriptor are used to establish the mapping between the global matrix and its corresponding process and distributed memory location.

In the Banded Linear Algebraic Equations sections throughout this book, the  $\_$  (underscore) symbol in the array descriptor is followed by an  $A$  or a  $B$ .  $A$  indicates a banded, tridiagonal, or symmetric tridiagonal matrix.  $B$  indicates a matrix containing the multiple right-hand sides matrix.

When you place a call to the banded or tridiagonal subroutines, you must be careful to choose consistent combinations of array descriptor types for matrix  $A$  and matrix  $B$ , and process grids. For consistent combinations, see the “Notes and Coding Rules” in the subroutine descriptions in Part 2 of this book.

Therefore, depending on which subroutine you are using in the Banded Linear Algebraic Equations, you may choose different array descriptors in the same subroutine calling sequence. Keep in mind **you must only create one process grid**; that is,  $CTXT\_A = CTXT\_B$ .

For example, when calling PDPBSV suppose you choose  $DTYPE\_A = 501$  for the band matrix  $A$  and  $DTYPE\_B = 502$  for matrix  $B$ . If you specify  $CTXT\_A$  as  $1 \times p$ , you must also specify  $CTXT\_B$  as  $1 \times p$ . Or if you specify  $CTXT\_A$  as  $p \times 1$ , you must also specify  $CTXT\_B$  as  $p \times 1$ . For an example of how to set the array descriptor values, see “Example” on page 494.

Table 26. Type-501 Array Descriptor

DESC_( )	Symbolic name	Value
1	DTYPE_	DTYPE_ = 501 for $1 \times p$ or $p \times 1$ , where $p$ is the number of processes in a process grid.
2	CTXT_	BLACS context in which the global matrix is defined. The BLACS process grid can be defined as $1 \times p$ or $p \times 1$ .  (See “Using the BLACS” on page 80.)
3	N_	Number of columns in the global matrix
4	NB_	Column block size.
5	CSRC_	The process column over which the first column of the global matrix is distributed

Table 26. Type-501 Array Descriptor (continued)

DESC_()	Symbolic name	Value
6	LLD_	Leading dimension of the local array. (See “Determining the Number of Rows or Columns in Your Local Arrays.”) This value may be different on each process. For the tridiagonal subroutines, this argument is ignored.
7	—	Reserved.

Table 27. Type-502 Array Descriptor

DESC_()	Symbolic name	Value
1	DTYPE_	DTYPE_ = 502 for $p \times 1$ or $1 \times p$ , where $p$ is the number of processes in a process grid.
2	CTXT_	BLACS context in which the global matrix is defined. The BLACS process grid can be defined as $1 \times p$ or $p \times 1$ .  (See “Using the BLACS” on page 80.)
3	M_	Number of rows in the global matrix
4	MB_	Row block size.
5	RSRC_	The process row over which the first row of the global matrix is distributed
6	LLD_	Leading dimension of the local array. (See “Determining the Number of Rows or Columns in Your Local Arrays.”) This value may be different on each process. For the tridiagonal subroutines, this argument is ignored for matrix $A$ .
7	—	Reserved.

## Determining the Number of Rows or Columns in Your Local Arrays

For local arrays described by type-501 array descriptor, the number of rows in the local matrix is always equal to the number of rows in the global matrix. The number of columns in the local array is determined as follows:

- For a  $1 \times q$  process grid:  
 $LOCq(N_) = NUMROC(N_, NB_, mycol, CSRC_, q)$
- For  $q \times 1$  process grid:  
 $LOCq(N_) = NUMROC(N_, NB_, myrow, CSRC_, q)$

where:

- $N_$  is the number of columns in the global matrix.
- $NB_$  is the column block size.
- $mycol$ , for a  $1 \times q$  process grid, is the process column index. See “Using the BLACS” on page 80.
- $myrow$ , for a  $q \times 1$  process grid, is the process row index. See “Using the BLACS” on page 80.
- $CSRC_$  is element 5 of type-501 array descriptor.
- $q$  is the number of columns in the process grid.

For local arrays described by type-502 array descriptor, the number of columns in the local matrix is always equal to the number of columns in the global matrix. The number of rows in the local array is determined as follows:

- For a  $p \times 1$  process grid:  

$$LOCp(M_) = NUMROC(M_, MB_, myrow, RSRC_, p)$$
- For a  $1 \times p$  process grid:  

$$LOCp(M_) = NUMROC(M_, MB_, mycol, RSRC_, p)$$

where:

- $M_$  is the number of rows in the global matrix.
- $MB_$  is the row block size.
- $myrow$ , for a  $p \times 1$  process grid, is the process row index. See “Using the BLACS” on page 80.
- $mycol$ , for a  $1 \times p$  process grid, is the process column index. See “Using the BLACS” on page 80.
- $RSRC_$  is element 5 of type-502 array descriptor.
- $p$  is the number of rows in the process grid.

## Distributing Data Structures

You must distribute your data before calling Parallel ESSL from your message passing program. This section shows how you how to distribute your data.

All the Parallel ESSL message passing subroutines, except the Banded Linear Algebraic Equations and Fourier transform subroutines, support block-cyclic distribution. The Banded Linear Algebraic Equations and the Fourier transform subroutines only support block distribution.

The following sections provide examples for distributing data over one- or two-dimensional process grids:

- “Vectors”
- “Matrices” on page 40
- “Specifying Sequences for the Fourier Transforms” on page 64

## Vectors

Parallel ESSL supports block-cyclic distribution for vectors over one- or two-dimensional process grids. A vector is distributed over a single row or column of the process grid, except for PDURNG. For PDURNG, vectors are distributed block-cyclically over the entire one- or two-dimensional process grid using row-major order, where the length  $n$  of the vector  $x$  must be evenly divisible by the available processes  $np$  multiplied by the block size  $nb$ . In other words,  $n/(np)(nb)$  must be an integer.

### Block-Cyclic Distribution over One-Dimensional Process Grids

This example shows how a global vector of length 24 with blocks of size 3 is distributed block-cyclically over one-dimensional process grids. Assume the following:

- $X = (8, 2, 3, 6, 5, 1, 9, 5, 3, 6, 2, 4, 10, 7, 4, 2, 8, 2, 8, 9, 2, 3, 11, 10)$

Global vector  $x$ :

$$\begin{matrix} B,D & 0 \\ & \left[ \begin{array}{c} 8 \\ 2 \end{array} \right] \\ 0 & \end{matrix}$$

1	3
	--
	6
	5
2	1
	--
	9
	5
3	3
	--
	6
	2
4	4
	--
	10
	7
5	4
	--
	2
	8
6	2
	--
	8
	9
7	2
	--
	3
	11
	10

Column-oriented,  $4 \times 1$  process grid:

B,D	0
-----	
0	$P_{00}$
4	
-----	
1	$P_{10}$
5	
-----	
2	$P_{20}$
6	
-----	
3	$P_{30}$
7	

Local arrays:

p,q	0
-----	
0	8
	2
	3
	10
	7
	4
-----	
1	6
	5
	1
	2
	8
	2
-----	
	9
	5
	3

2	8
	9
	2
-----	
	6
	2
	4
3	3
	11
	10

For the column-oriented example, the array descriptor DESC\_X contains the following:

DESC_X()	Symbolic name	Value
1	DTYPE_X	1
2	CTXT_X	BLACS context
3	M_X	24
4	N_X	1
5	MB_X	3
6	NB_X	1
7	RSRC_X	0
8	CSRC_X	0
9	LLD_X	6

Row-oriented,  $1 \times 4$  process grid:

B,D	0 4	1 5	2 6	3 7
-----	-----	-----	-----	-----
0	P <sub>00</sub>	P <sub>01</sub>	P <sub>02</sub>	P <sub>03</sub>

Local array:

p,q	0	1	2	3
-----	-----	-----	-----	-----
0	8 2 3 10 7 4	6 5 1 2 8 2	9 5 3 8 9 2	6 2 4 3 11 10

For the row-oriented example, the array descriptor DESC\_X contains the following:

DESC_X()	Symbolic name	Value
1	DTYPE_X	1
2	CTXT_X	BLACS context
3	M_X	1
4	N_X	24
5	MB_X	1
6	NB_X	3
7	RSRC_X	0
8	CSRC_X	0
9	LLD_X	1

**Note:** The same global vector was distributed over a  $4 \times 1$  grid and then over a  $1 \times 4$  grid. Notice the values contained in the corresponding local arrays are identical.

## Block-Cyclic Distribution over Two-Dimensional Process Grids

This example shows how a global vector of length 18 with block size of 3 is distributed over two-dimensional grids. When a two-dimensional process grid is used, the global vector can be distributed over any single row or any single column of the grid. Assume the following:

- $X = (4, 11, 17, 21, 3, 7, 12, 5, 3, 15, 3, 4, 9, 17, 1, 10, 9, 25)$

Global vector  $x$ :

B,D	0
0	4
	11
	17
	--
	21
1	3
	7
	--
	12
2	5
	3
	--
	15
3	3
	4
	--
	9
4	17
	1
	--
	10
5	9
	25

Two-dimensional,  $2 \times 3$  process grid:

B,D	--	--	0
0	P <sub>00</sub>	P <sub>01</sub>	P <sub>02</sub>
2			
4			
1	P <sub>10</sub>	P <sub>11</sub>	P <sub>12</sub>
3			
5			

If the global vector is distributed over the third column of a  $2 \times 3$  process grid, then P<sub>02</sub> and P<sub>12</sub> contain the following local arrays:

p,q	2
0	4
	11
	17
	12
	5
	3
	9
	17
	1
	21
	3
	7

1	15
	3
	4
	10
	9
	25

For the single column example, the array descriptor DESC\_X contains the following:

DESC_X( )	Symbolic name	Value
1	DTYPE_X	1
2	CTXT_X	BLACS context
3	M_X	18
4	N_X	1
5	MB_X	3
6	NB_X	1
7	RSRC_X	0
8	CSRC_X	2
9	LLD_X	9

If the global vector is distributed over the second row of a  $2 \times 3$  process grid, then  $P_{10}$ ,  $P_{11}$ , and  $P_{12}$  contain the following local arrays:

p,q	0	1	2
1	4 11 17 15 3 4	21 3 7 9 17 1	12 5 3 10 9 25

For the single row example, the array descriptor DESC\_X contains the following:

DESC_X( )	Symbolic name	Value
1	DTYPE_X	1
2	CTXT_X	BLACS context
3	M_X	1
4	N_X	18
5	MB_X	1
6	NB_X	3
7	RSRC_X	1
8	CSRC_X	0
9	LLD_X	1

For PDURNG, the global vector is distributed block-cyclically over the **entire**  $2 \times 3$  process grid using row-major order, as follows:

p,q	0	1	2
0	4 11 17	21 3 7	12 5 3
1	15 3 4	9 17 1	10 9 25

**Notes:**

1. For PDURNG, the length  $n$  of the vector  $x$  must be evenly divisible by the number of available processes  $np$  multiplied by the block size  $nb$ . For this example,  $18 = (6)(3)$ .
2. For PDURNG, the array descriptor is not used.

Following is an example of uneven block-cyclic distribution for a global vector of length 20 with block size of 3, where the two local arrays are different sizes. In this case, a fragment of a block with two elements occurs at the end of the vector.

Assume the following:

$x = (0, 5, 6, 3, 21, 5, 6, 1, 8, 9, 13, 11, 12, 15, 14, 15, 11, 17, 18, 19)$

Following is a global vector  $x$  with block size 3:

B,D	0
0	0
	5
	6
1	--
	3
	21
2	5
	--
	6
3	1
	8
	--
4	9
	13
	11
5	--
	12
	15
6	14
	--
	15
	11
	17
	--
	18
	19

Two-dimensional,  $2 \times 3$  process grid:

B,D	0	--	--
0	P <sub>00</sub>	P <sub>01</sub>	P <sub>02</sub>
2			
4			
6			
1	P <sub>10</sub>	P <sub>11</sub>	P <sub>12</sub>
3			
5			

If the vector is distributed over the first column of a  $2 \times 3$  process grid, then P<sub>00</sub> and P<sub>10</sub> contain the following local arrays:

p,q	0
0	0
	5
	6



0	6
	1
	8
	12
	15
	14
	18
	19
-----	
1	3
	21
	5
	9
	13
	11
	15
	11
	17

Array descriptor DESC\_X contains the following:

DESC_X( )	Symbolic name	Value
1	DTYPE_X	1
2	CTXT_X	BLACS context
3	M_X	20
4	N_X	1
5	MB_X	3
6	NB_X	1
7	RSRC_X	0
8	CSRC_X	0
9	LLD_X	11 (For P <sub>00</sub> ) 9 (For P <sub>10</sub> )

If the vector is distributed over the first row of the  $2 \times 3$  process grid, then P<sub>00</sub>, P<sub>01</sub>, and P<sub>02</sub> contain the following local arrays:

p,q	0	1	2
0	0 5 6 9 13 11 18 19	3 21 5 12 15 14	6 1 8 15 11 17

Array descriptor DESC\_X contains the following:

DESC_X( )	Symbolic name	Value
1	DTYPE_X	1
2	CTXT_X	BLACS context
3	M_X	1
4	N_X	20
5	MB_X	1
6	NB_X	3
7	RSRC_X	0
8	CSRC_X	0
9	LLD_X	1

## Matrices

The Parallel ESSL subroutines, except the Banded Linear Algebraic Equations, support block-cyclic data distribution for matrices using one- or two-dimensional process grids. The Banded Linear Algebraic Equations support only block data distribution using one-dimensional process grids.

The following terminology is used when it is necessary to distinguish special types of matrices:

- Full block matrix — a matrix of blocks distributed over the whole process grid.
- Block row matrix — a matrix of blocks distributed over a single row of the process grid.
- Block column matrix — a matrix of blocks distributed over a single column of the process grid.
- Single block matrix — a matrix consisting of a single block lying in a single process of the process grid.

### Distributed over One-Dimensional Process Grids

This section describes how to distribute a matrix block-cyclically over a one-dimensional process grid. It also shows how matrices for the Banded Linear Algebraic Equations are distributed over a one-dimensional process grid using block distribution.

**Block-Cyclically Distributing a Matrix:** The examples that follow show how a  $6 \times 8$  global matrix  $A$  with blocks of size  $2 \times 2$  is distributed block-cyclically over one-dimensional process grids. Assume the following global matrix  $A$ :

B,D	0	1	2	3
0	$\begin{bmatrix} 0 & 1 \\ 10 & 11 \end{bmatrix}$	$\begin{bmatrix} 2 & 3 \\ 12 & 13 \end{bmatrix}$	$\begin{bmatrix} 4 & 5 \\ 14 & 15 \end{bmatrix}$	$\begin{bmatrix} 6 & 7 \\ 16 & 17 \end{bmatrix}$
1	$\begin{bmatrix} 20 & 21 \\ 30 & 31 \end{bmatrix}$	$\begin{bmatrix} 22 & 23 \\ 32 & 33 \end{bmatrix}$	$\begin{bmatrix} 24 & 25 \\ 34 & 35 \end{bmatrix}$	$\begin{bmatrix} 26 & 27 \\ 36 & 37 \end{bmatrix}$
2	$\begin{bmatrix} 40 & 41 \\ 50 & 51 \end{bmatrix}$	$\begin{bmatrix} 42 & 43 \\ 52 & 53 \end{bmatrix}$	$\begin{bmatrix} 44 & 45 \\ 54 & 55 \end{bmatrix}$	$\begin{bmatrix} 46 & 47 \\ 56 & 57 \end{bmatrix}$

Column-oriented,  $3 \times 1$  process grid:

B,D	0 1 2 3
0	$P_{00}$
1	$P_{10}$
2	$P_{20}$

Local arrays:

p,q	0
0	$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 \end{bmatrix}$
1	$\begin{bmatrix} 20 & 21 & 22 & 23 & 24 & 25 & 26 & 27 \\ 30 & 31 & 32 & 33 & 34 & 35 & 36 & 37 \end{bmatrix}$
2	$\begin{bmatrix} 40 & 41 & 42 & 43 & 44 & 45 & 46 & 47 \\ 50 & 51 & 52 & 53 & 54 & 55 & 56 & 57 \end{bmatrix}$

For the column-oriented example, the array descriptor DESC\_A contains:

DESC_A( )	Symbolic name	Value
1	DTYPE_A	1
2	CTXT_A	BLACS context
3	M_A	6
4	N_A	8
5	MB_A	2
6	NB_A	2
7	RSRC_A	0
8	CSRC_A	0
9	LLD_A	2

Row-oriented,  $1 \times 2$  process grid:

B,D	0 2	1 3
0	P <sub>00</sub>	P <sub>01</sub>
1		
2		

Local arrays:

p,q	0	1
	0 1 4 5	2 3 6 7
	10 11 14 15	12 13 16 17
	20 21 24 25	22 23 26 27
0	30 31 34 35	32 33 36 37
	40 41 44 45	42 43 46 47
	50 51 54 55	52 53 56 57

For the row-oriented example, the array descriptor DESC\_A:

DESC_A( )	Symbolic name	Value
1	DTYPE_A	1
2	CTXT_A	BLACS context
3	M_A	6
4	N_A	8
5	MB_A	2
6	NB_A	2
7	RSRC_A	0
8	CSRC_A	0
9	LLD_A	6

For an example of distributing a matrix over a one-dimensional process grid in a Fortran 90 program, see matrix *F* in Appendix B, “Sample Programs,” which is:

- Created in subroutine initialize\_carray in “Module Scale” on page 919.
- Assigned values in subroutine get\_diffusion\_matrix in “Module Fourier” on page 911.
- Used in “Program Main” on page 902.

**Block-Cyclically Distributing a Symmetric Band Matrix:** This section shows how to distribute a symmetric band matrix  $A$  over a one-dimensional process grid using block-cyclic distribution.

Assume the following symmetric band matrix  $A$  of size  $9 \times 9$  with a half bandwidth of 2:

$$A = \begin{bmatrix} 11 & 21 & 31 & 0 & 0 & 0 & 0 & 0 & 0 \\ 21 & 22 & 32 & 42 & 0 & 0 & 0 & 0 & 0 \\ 31 & 32 & 33 & 34 & 53 & 0 & 0 & 0 & 0 \\ 0 & 42 & 34 & 44 & 54 & 64 & 0 & 0 & 0 \\ 0 & 0 & 53 & 54 & 55 & 65 & 75 & 0 & 0 \\ 0 & 0 & 0 & 64 & 65 & 66 & 76 & 86 & 0 \\ 0 & 0 & 0 & 0 & 75 & 76 & 77 & 87 & 97 \\ 0 & 0 & 0 & 0 & 0 & 86 & 87 & 88 & 98 \\ 0 & 0 & 0 & 0 & 0 & 0 & 97 & 98 & 99 \end{bmatrix}$$

Matrix  $A$  must be stored in upper- or lower-band-packed storage mode. The sections that follow contain examples describing these two storage modes. In these examples, matrix  $A$  is stored in an array with dimensions  $3 \times 9$ .

*Upper-Band-Packed Storage Mode:* The global matrix  $A$  with block size of 2 is stored in upper-band-packed storage mode, as follows:

B,D	0	1	2	3	4
0	$\begin{bmatrix} * & * \\ * & 21 \\ 11 & 22 \end{bmatrix}$	$\begin{bmatrix} 31 & 42 \\ 32 & 34 \\ 33 & 44 \end{bmatrix}$	$\begin{bmatrix} 53 & 64 \\ 54 & 65 \\ 55 & 66 \end{bmatrix}$	$\begin{bmatrix} 75 & 86 \\ 76 & 87 \\ 77 & 88 \end{bmatrix}$	$\begin{bmatrix} 97 \\ 98 \\ 99 \end{bmatrix}$

Following is a row-oriented,  $1 \times 3$  process grid:

B,D	0 3	1 4	2
0	P <sub>00</sub>	P <sub>01</sub>	P <sub>02</sub>

The following local arrays  $A$  are distributed block-cyclically over the  $1 \times 3$  process grid:

p,q	0	1	2
0	$\begin{bmatrix} * & * & 75 & 86 \\ * & 21 & 76 & 87 \\ 11 & 22 & 77 & 88 \end{bmatrix}$	$\begin{bmatrix} 31 & 42 & 97 \\ 32 & 34 & 98 \\ 33 & 44 & 99 \end{bmatrix}$	$\begin{bmatrix} 53 & 64 \\ 54 & 65 \\ 55 & 66 \end{bmatrix}$

where  $*$  means you do not have to store a value in that position in the local array. However, these storage positions are required and overwritten during the computation.

The type-501 array descriptor DESC\_A contains the following:

DESC_A( )	Symbolic name	Value
1	DTYPE_A	DTYPE_A = 501 for $1 \times p$
2	CTXT_A	BLACS context
3	N_A	9
4	NB_A	2
5	CSRC_A	0
6	LLD_A	3
7	—	Reserved

Alternately, the type-1 array descriptor DESC\_A contains the following:

DESC_A( )	Symbolic name	Value
1	DTYPE_A	DTYPE_A = 1 for $1 \times p$
2	CTXT_A	BLACS context
3	M_A	3
4	N_A	9
5	MB_A	1
6	NB_A	2
7	RSRC_A	0
8	CSRC_A	0
9	LLD_A	3

*Lower-Band-Packed Storage Mode:* The global matrix  $A$  with block size of 2 is stored in lower-band-packed storage mode, as follows:

$$\begin{array}{c}
 \text{B,D} \quad \quad 0 \quad \quad 1 \quad \quad 2 \quad \quad 3 \quad \quad 4 \\
 \\
 0 \quad \left[ \begin{array}{cc|cc|cc|cc|cc}
 11 & 22 & 33 & 44 & 55 & 66 & 77 & 88 & 99 & \\
 21 & 32 & 34 & 54 & 65 & 76 & 87 & 98 & * & \\
 31 & 42 & 53 & 64 & 75 & 86 & 97 & * & * & 
 \end{array} \right]
 \end{array}$$

Following is a row-oriented,  $1 \times 3$  process grid:

$$\begin{array}{c}
 \text{B,D} \quad \left| \begin{array}{cc} 0 & 3 \end{array} \right| \quad \left| \begin{array}{cc} 1 & 4 \end{array} \right| \quad \left| \begin{array}{cc} 2 & \end{array} \right| \\
 \hline
 0 \quad \left| \begin{array}{cc} \text{P}_{00} & \end{array} \right| \quad \left| \begin{array}{cc} \text{P}_{01} & \end{array} \right| \quad \left| \begin{array}{cc} \text{P}_{02} & \end{array} \right|
 \end{array}$$

The following local arrays  $A$  are distributed block-cyclically over the  $1 \times 3$  process grid:

$$\begin{array}{c}
 \text{p,q} \quad \left| \begin{array}{cccc} 0 & & & \end{array} \right| \quad \left| \begin{array}{cccc} 1 & & & \end{array} \right| \quad \left| \begin{array}{cccc} 2 & & & \end{array} \right| \\
 \hline
 0 \quad \left| \begin{array}{cccc}
 11 & 22 & 77 & 88 \\
 21 & 32 & 87 & 98 \\
 31 & 42 & 97 & * \\
 \end{array} \right| \quad \left| \begin{array}{cccc}
 33 & 44 & 99 & \\
 34 & 54 & * & \\
 53 & 64 & * & 
 \end{array} \right| \quad \left| \begin{array}{cccc}
 55 & 66 & & \\
 65 & 76 & & \\
 75 & 86 & & 
 \end{array} \right|
 \end{array}$$

where \* means you do not have to store a value in that position in the local array. However, these storage positions are required and overwritten during the computation.

The type-501 array descriptor DESC\_A contains the following:

DESC_A( )	Symbolic name	Value
1	DTYPE_A	DTYPE_A = 501 for $1 \times p$
2	CTXT_A	BLACS context
3	N_A	9
4	NB_A	2
5	CSRC_A	0
6	LLD_A	3
7	—	Reserved

Alternately, the type-1 array descriptor DESC\_A contains the following:

DESC_A( )	Symbolic name	Value
1	DTYPE_A	DTYPE_A = 1 for $1 \times p$
2	CTXT_A	BLACS context
3	M_A	3
4	N_A	9
5	MB_A	1
6	NB_A	2
7	RSRC_A	0
8	CSRC_A	0
9	LLD_A	3

For more information on how to store symmetric band matrices, see the *ESSL Version 3 Release 1.1 Guide and Reference* manual.

**Block-Cyclically Distributing a General Tridiagonal Matrix:** A general tridiagonal matrix, represented as three vectors, must be distributed over a one-dimensional process grid using a block-cyclic data distribution. Because vectors are one-dimensional data structures, you can use a type-501, type-502, or type-1 array descriptor regardless of whether the process grid is  $1 \times p$  or  $p \times 1$ .

The first part of this section shows how to distribute a general tridiagonal matrix  $A$  over a  $p \times 1$  process grid. The second part shows how to distribute the same matrix over a  $1 \times p$  process grid. **In both cases, the values contained in the corresponding local arrays are identical.**

Assume the following general tridiagonal matrix  $A$  of size  $7 \times 7$ :

$$\begin{bmatrix} 11 & 12 & 0 & 0 & 0 & 0 & 0 \\ 21 & 22 & 23 & 0 & 0 & 0 & 0 \\ 0 & 32 & 33 & 34 & 0 & 0 & 0 \\ 0 & 0 & 43 & 44 & 45 & 0 & 0 \\ 0 & 0 & 0 & 54 & 55 & 56 & 0 \\ 0 & 0 & 0 & 0 & 65 & 66 & 67 \\ 0 & 0 & 0 & 0 & 0 & 76 & 77 \end{bmatrix}$$

Matrix  $A$  is stored in tridiagonal storage mode in the following three vectors:

$dl = (*, 21, 32, 43, 54, 65, 76)$

$d = (11, 22, 33, 44, 55, 66, 77)$

$du = (12, 23, 34, 45, 56, 67, *)$

*Block-Cyclic Distribution on a  $p \times 1$  Process Grid:* The general tridiagonal matrix  $A$  is stored in tridiagonal storage mode in vectors  $dl$ ,  $d$ , and  $du$ .

Following is global vector  $dl$ :

B,D	0	
0		$\begin{bmatrix} * \\ 21 \\ -- \\ 32 \\ 43 \\ -- \\ 54 \\ 65 \\ -- \\ 76 \end{bmatrix}$
1		
2		
3		

Following is global vector  $d$ :

B,D	0	
0		$\begin{bmatrix} 11 \\ 22 \\ -- \\ 33 \\ 44 \\ -- \\ 55 \\ 66 \\ -- \\ 77 \end{bmatrix}$
1		
2		
3		

Following is global vector  $du$ :

B,D	0	
0		$\begin{bmatrix} 12 \\ 23 \\ -- \\ 34 \\ 45 \\ -- \\ 56 \\ 67 \\ -- \\ * \end{bmatrix}$
1		
2		
3		

Following is a column-oriented,  $3 \times 1$  process grid:

B,D	0
0	$P_{00}$
3	
1	$P_{10}$
2	$P_{20}$

The arrays are block-cyclically distributed over the  $3 \times 1$  process grid.

Following are the local arrays for DL:

p,q	0
0	* 21 76
1	32 43
2	54 65

Following are the local arrays for D:

p,q	0
0	11 22 77
1	33 44
2	55 66

Following are the local arrays for DU:

p,q	0
0	12 23 *
1	34 45
2	56 67

where “\*” means you do not have to store a value in that position in the local array. However, these storage positions are required.

The type-502 array descriptor DESC\_A contains the following:

DESC_A( )	Symbolic name	Value
1	DTYPE_A	DTYPE_A = 502 for $p \times 1$
2	CTXT_A	BLACS context
3	M_A	7
4	MB_A	2
5	RSRC_A	0
6	LLD_A	Not used
7	–	Reserved



Alternately, the type-1 array descriptor DESC\_A contains the following:

DESC_A( )	Symbolic name	Value
1	DTYPE_A	DTYPE_A = 1 for $p \times 1$
2	CTXT_A	BLACS context
3	M_A	7
4	N_A	1
5	MB_A	2
6	NB_A	1
7	RSRC_A	0
8	CSRC_A	0
9	LLD_A	Not used

*Block-Cyclic Distribution on a  $1 \times p$  Process Grid:* The general tridiagonal matrix  $A$  is stored in tridiagonal storage mode in vectors  $dl$ ,  $d$ , and  $du$ . Because vectors are one-dimensional data structures, the block-cyclically distributed arrays on a  $1 \times p$  process grid are identical to the block-cyclically distributed arrays on a  $p \times 1$  process grid.

Following is global vector  $dl$ :

B,D      0          1          2          3  
0     $\left[ \begin{array}{c|c|c|c} * & 21 & 32 & 43 & 54 & 65 & 76 \end{array} \right]$

Following is global vector  $d$ :

B,D      0          1          2          3  
0     $\left[ \begin{array}{c|c|c|c} 11 & 22 & 33 & 44 & 55 & 66 & 77 \end{array} \right]$

Following is global vectors  $du$ :

B,D      0          1          2          3  
0     $\left[ \begin{array}{c|c|c|c} 12 & 23 & 34 & 45 & 55 & 67 & * \end{array} \right]$

Following is a row-oriented,  $1 \times 3$  process grid:

B,D     $\left| \begin{array}{c|c|c} 0 & 3 & 1 & 2 \\ \hline 0 & P_{00} & P_{01} & P_{02} \end{array} \right|$

The arrays are block-cyclically distributed over the  $1 \times 3$  process grid.

Following are the local arrays for DL:

p,q     $\left| \begin{array}{c|c|c} 0 & 1 & 2 \\ \hline 0 & * & 21 & 76 & 32 & 43 & 54 & 65 \end{array} \right|$

Following are the local arrays for D:

p,q     $\left| \begin{array}{c|c|c} 0 & 1 & 2 \\ \hline 0 & 11 & 22 & 77 & 33 & 44 & 55 & 66 \end{array} \right|$

Following are the local arrays for DU:

p,q	0	1	2
0	12 23 *	34 45	55 67

where “\*” means you do not have to store a value in that position in the local array. However, these storage positions are required.

The type-501 array descriptor DESC\_A contains the following:

DESC_A( )	Symbolic name	Value
1	DTYPE_A	DTYPE_A = 501 for $1 \times p$
2	CTXT_A	BLACS context
3	N_A	7
4	NB_A	2
5	CSRC_A	0
6	LLD_A	Not used
7	–	Reserved

Alternately, the type-1 array descriptor DESC\_A contains the following:

DESC_A( )	Symbolic name	Value
1	DTYPE_A	DTYPE_A = 1 for $1 \times p$
2	CTXT_A	BLACS context
3	M_A	1
4	N_A	7
5	MB_A	1
6	NB_A	2
7	RSRC_A	0
8	CSRC_A	0
9	LLD_A	Not used

For more information on how to store general tridiagonal matrices, see the *ESSL Version 3 Release 1.1 Guide and Reference* manual.

**Block-Cyclically Distributing a Symmetric Tridiagonal Matrix:** A symmetric tridiagonal matrix, represented as two vectors, must be distributed over a one-dimensional process grid using a block-cyclic data distribution. Because vectors are one-dimensional data structures, you can use a type-501, type-502, or type-1 array descriptor regardless of whether the process grid is  $p \times 1$  or  $1 \times p$ .

**Note:** For both serial ESSL and Parallel ESSL, the  $n-1$  elements of the equal off-diagonals of a symmetric tridiagonal matrix are stored in a one-dimensional vector of length  $n$ . To be compatible with ScaLAPACK, in Parallel ESSL, the off-diagonal is chosen to be the superdiagonal and is stored in elements 1 through  $n-1$ . In the serial ESSL library, the off-diagonal is chosen to be the subdiagonal and is stored in elements 2 through  $n$ .

The first part of this section shows a how to distribute a symmetric tridiagonal matrix  $A$  over a  $p \times 1$  process grid. The second part shows how to distribute the same matrix over a  $1 \times p$  process grid. **In both cases, the values contained in the corresponding local arrays are identical.**

Assume the following symmetric tridiagonal matrix  $A$  of size  $7 \times 7$ :

$$\begin{bmatrix} 10 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 20 & 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 30 & 3 & 0 & 0 & 0 \\ 0 & 0 & 3 & 40 & 4 & 0 & 0 \\ 0 & 0 & 0 & 4 & 50 & 5 & 0 \\ 0 & 0 & 0 & 0 & 5 & 60 & 6 \\ 0 & 0 & 0 & 0 & 0 & 6 & 70 \end{bmatrix}$$

Matrix  $A$  is stored in parallel-symmetric-tridiagonal storage mode in the following two vectors:

$$d = (10, 20, 30, 40, 50, 60, 70)$$

$$e = (1, 2, 3, 4, 5, 6, *)$$

*Block-Cyclic Distribution on a  $p \times 1$  Process Grid:* The symmetric tridiagonal matrix  $A$  is stored in parallel-symmetric-tridiagonal storage mode in vectors  $d$  and  $e$ .

Following is global vector  $d$ :

$$\begin{array}{c|c} \text{B,D} & 0 \\ \hline 0 & \begin{bmatrix} 10 \\ 20 \\ 30 \\ -- \\ 40 \\ 50 \\ 60 \\ -- \\ 70 \end{bmatrix} \\ 1 & \\ 2 & \end{array}$$

Following is global vector  $e$ :

$$\begin{array}{c|c} \text{B,D} & 0 \\ \hline 0 & \begin{bmatrix} 1 \\ 2 \\ 3 \\ - \\ 4 \\ 5 \\ 6 \\ - \\ * \end{bmatrix} \\ 1 & \\ 2 & \end{array}$$

Following is a column-oriented,  $2 \times 1$  process grid:

$$\begin{array}{c|c} \text{B,D} & 0 \\ \hline 0 & P_{00} \\ 2 & \\ \hline 1 & P_{10} \end{array}$$

The arrays are block-cyclically distributed over the  $2 \times 1$  process grid.

Following are the local arrays for D:

p,q	0
-----	
0	10
	20
	30
	70
-----	
1	40
	50
	60

Following are the local arrays for E:

p,q	0
-----	
0	1
	2
	3
	*
-----	
1	4
	5
	6

where \* means you do not have to store a value in that position in the local array. However, these storage positions are required.

The type-502 array descriptor DESC\_A contains the following:

DESC_A( )	Symbolic name	Value
1	DTYPE_A	DTYPE_A = 502 for $p \times 1$
2	CTXT_A	BLACS context
3	M_A	7
4	MB_A	3
5	RSRC_A	0
6	LLD_A	Not used
7	–	Reserved

Alternately, the type-1 array descriptor DESC\_A contains the following:

DESC_A( )	Symbolic name	Value
1	DTYPE_A	DTYPE_A = 1 for $p \times 1$
2	CTXT_A	BLACS context
3	M_A	7
4	N_A	1
5	MB_A	3
6	NB_A	1
7	RSRC_A	0
8	CSRC_A	0
9	LLD_A	Not used

*Block-Cyclic Distribution on a  $1 \times p$  Process Grid:* The symmetric tridiagonal matrix  $A$  is stored in parallel-symmetric-tridiagonal storage mode in vectors  $d$  and  $e$ . Because vectors are one-dimensional data structures, the block-cyclically distributed arrays on a  $1 \times p$  process grid are identical to the block-cyclically distributed arrays on a  $p \times 1$  process grid.

Following is global vector  $d$ :

B,D                      0                      1                      2  
 0     $\left[ \begin{array}{ccc|ccc|ccc} 10 & 20 & 30 & 40 & 50 & 60 & 70 \end{array} \right]$

Following is global vector  $e$ :

B,D                      0                      1                      2  
 0     $\left[ \begin{array}{ccc|ccc|c} 1 & 2 & 3 & 4 & 5 & 6 & * \end{array} \right]$

Following is a row-oriented,  $1 \times 2$  process grid:

B,D     $\left| \begin{array}{cc} 0 & 2 \end{array} \right| \left| \begin{array}{c} 1 \end{array} \right|$   
 -----  
 0     $\left| \begin{array}{cc} P_{00} \end{array} \right| \left| \begin{array}{c} P_{01} \end{array} \right|$

The arrays are block-cyclically distributed over the  $1 \times 2$  process grid.

Following are the local arrays for D:

p,q     $\left| \begin{array}{ccc} 0 \end{array} \right| \left| \begin{array}{ccc} 1 \end{array} \right|$   
 -----  
 0     $\left| \begin{array}{cccc} 10 & 20 & 30 & 70 \end{array} \right| \left| \begin{array}{ccc} 40 & 50 & 60 \end{array} \right|$

Following are the local arrays for E:

p,q     $\left| \begin{array}{ccc} 0 \end{array} \right| \left| \begin{array}{ccc} 1 \end{array} \right|$   
 -----  
 0     $\left| \begin{array}{cccc} 1 & 2 & 3 & * \end{array} \right| \left| \begin{array}{ccc} 4 & 5 & 6 \end{array} \right|$

where “\*” means you do not have to store a value in that position in the local array. However, these storage positions are required.

The type-501 array descriptor DESC\_A contains the following:

DESC_A ( )	Symbolic name	Value
1	DTYPE_A	DTYPE_A = 501 for $1 \times p$
2	CTXT_A	BLACS context
3	N_A	7
4	NB_A	3
5	CSRC_A	0
6	LLD_A	Not used
7	–	Reserved

Alternately, the type-1 array descriptor DESC\_A contains the following:

DESC_A( )	Symbolic name	Value
1	DTYPE_A	DTYPE_A = 1 for $1 \times p$
2	CTXT_A	BLACS context
3	M_A	1
4	N_A	7
5	MB_A	1
6	NB_A	3
7	RSRC_A	0
8	CSRC_A	0
9	LLD_A	Not used

### Block-Cyclically Distributing a General Matrix Containing the Right-Hand

**Sides:** This section shows how to block-cyclically distribute a general matrix  $B$  containing the multiple right-hand sides for the Banded Linear Algebraic Equations subroutines.

Following is the global matrix  $B$ :

B,D	0
0	$\begin{bmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ \hline 31 & 32 & 33 \\ 41 & 42 & 43 \\ \hline 51 & 52 & 53 \\ 61 & 62 & 63 \\ \hline 71 & 72 & 73 \end{bmatrix}$
1	
2	
3	

Following is a  $3 \times 1$  process grid:

B,D	0
0	$P_{00}$
3	
1	$P_{10}$
2	$P_{20}$

Following are the local arrays:

p,q	0
0	$\begin{bmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 71 & 72 & 73 \end{bmatrix}$
1	$\begin{bmatrix} 31 & 32 & 33 \\ 41 & 42 & 43 \end{bmatrix}$
2	$\begin{bmatrix} 51 & 52 & 53 \\ 61 & 62 & 63 \end{bmatrix}$

The type-502 array descriptor DESC\_B contains the following:

DESC_B( )	Symbolic name	Value
1	DTYPE_B	DTYPE_B = 502 for $p \times 1$
2	CTXT_B	BLACS context
3	M_B	7
4	MB_B	2
5	RSRC_B	0
6	LLD_B	3 (For $P_{00}$ ) 2 (For $P_{10}$ and $P_{20}$ )
7	—	Reserved

Alternately, the type-1 array descriptor DESC\_B contains the following:

DESC_B( )	Symbolic name	Value
1	DTYPE_B	DTYPE_B = 1 for $p \times 1$
2	CTXT_B	BLACS context
3	M_B	7
4	N_B	3
5	MB_B	2
6	NB_B	1
7	RSRC_B	0
8	CSRC_B	0
9	LLD_B	3 (For $P_{00}$ ) 2 (For $P_{10}$ and $P_{20}$ )

## Block-Cyclically Distributing over Two-Dimensional Process Grids

This section shows how to distribute general, symmetric, and upper triangular matrices over a two-dimensional process grid using block-cyclic distribution.

**Distributing a General Matrix:** This example shows how the data for a global matrix  $A$  with block size of  $2 \times 3$  is distributed block-cyclically over the entire  $2 \times 3$  process grid. Assume the following  $9 \times 26$  global matrix  $A$  with 45 blocks:

B,D	0	1	2	3	4	5	6	7	8
0	112 5 7 116 9 6	8 9 3 7 2 3	7 5 1 6 5 6	3 2 1 4 3 2	8 98 4 7 2 111	8 9 4 7 2 1	1 3 10 7 6 15	3 3 10 7 6 15	5 3 7 6
1	1 5 7 6 9 6	1 9 3 7 2 3	1 5 1 6 5 6	1 2 1 4 3 2	1 9 4 7 2 1	1 9 4 7 2 1	5 8 10 7 6 19	3 3 11 7 1 15	5 3 7 2
2	2 5 7 6 9 6	2 9 3 7 2 3	2 5 1 6 5 6	2 2 1 4 3 2	2 9 4 7 2 1	2 9 4 7 2 1	1 8 10 7 3 19	2 3 11 7 4 15	3 3 7 8
3	3 5 7 6 9 6	3 9 3 7 2 3	3 5 1 6 5 6	3 2 1 4 3 2	3 9 4 7 2 1	3 9 4 7 2 1	9 8 10 1 3 49	2 3 11 7 4 55	3 3 7 3
4	20 1 9	4 5 6	9 8 7	1 4 3	1 15 21	4 7 6	9 8 12	3 9 18	2 4

**Note:** In this example, the global matrix dimensions are not divisible by the respective block size. As a result, all of the block sizes are  $2 \times 3$ , except for blocks in the last row and the last column of the blocked matrix.

Two-dimensional,  $2 \times 3$  process grid:

B,D	0 3 6	1 4 7	2 5 8
0	P <sub>00</sub>	P <sub>01</sub>	P <sub>02</sub>
2			
4			
1	P <sub>10</sub>	P <sub>11</sub>	P <sub>12</sub>
3			

Local arrays:

p,q	0	1	2
0	112 5 7 3 2 1 1 3 10 116 9 6 4 3 2 7 6 15 2 5 7 2 2 1 1 8 10 6 9 6 4 3 2 7 3 19 20 1 9 1 4 3 9 8 12	8 9 3 8 98 4 3 3 10 7 2 3 7 2 111 7 6 15 2 9 3 2 9 4 2 3 11 7 2 3 7 2 1 7 4 15 4 5 6 1 15 21 3 9 18	7 5 1 8 9 4 5 3 6 5 6 7 2 1 7 6 2 5 1 2 9 4 3 3 6 5 6 7 2 1 7 8 9 8 7 4 7 6 2 4
1	1 5 7 1 2 1 5 8 10 6 9 6 4 3 2 7 6 19 3 5 7 3 2 1 9 8 10 6 9 6 4 3 2 1 3 49	1 9 3 1 9 4 3 3 11 7 2 3 7 2 1 7 1 15 3 9 3 3 9 4 2 3 11 7 2 3 7 2 1 7 4 55	1 5 1 1 9 4 5 3 6 5 6 7 2 1 7 2 3 5 1 3 9 4 3 3 6 5 6 7 2 1 7 3



Array descriptor DESC\_A contains the following:

DESC_A( )	Symbolic name	Value
1	DTYPE_A	1
2	CTXT_A	BLACS context
3	M_A	9
4	N_A	26
5	MB_A	2
6	NB_A	3
7	RSRC_A	0
8	CSRC_A	0
9	LLD_A	5 (For P <sub>00</sub> , P <sub>01</sub> , and P <sub>02</sub> ) 4 (For P <sub>10</sub> , P <sub>11</sub> , and P <sub>12</sub> )

**Distributing a Symmetric Matrix:** This example shows how the data for a global symmetric matrix *A* with block size of  $3 \times 3$  is distributed block-cyclically over a  $2 \times 3$  process grid. Assume the following  $18 \times 18$  global symmetric matrix *A* with 36 blocks:

B,D	0	1	2	3	4	5
0	1 2 3 2 10 11 3 11 20	4 5 6 12 13 14 21 22 23	7 8 9 15 16 17 24 25 26	10 11 12 18 19 20 27 28 29	13 14 15 21 22 23 30 31 32	16 17 18 24 25 26 33 34 35
1	4 12 21 5 13 22 6 14 23	2 3 5 3 1 4 5 4 5	7 11 13 9 16 25 6 10 11	17 19 23 36 49 64 15 16 20	29 31 37 81 10 12 21 25 26	41 43 47 14 16 19 30 31 35
2	7 15 24 8 16 25 9 17 26	7 9 6 11 16 10 13 25 11	1 2 3 2 11 13 3 13 2	4 5 6 15 17 19 4 6 8	7 8 9 21 23 25 10 12 14	10 11 12 27 29 31 16 18 20
3	10 18 27 11 19 28 12 20 29	17 36 15 19 49 16 23 64 20	4 15 4 5 17 6 6 19 8	3 6 9 6 1 2 9 2 1	2 4 6 3 4 5 3 5 7	3 6 9 6 7 8 9 11 13
4	13 21 30 14 22 31 15 23 32	29 81 21 31 10 25 37 12 26	7 21 10 8 23 12 9 25 14	2 3 3 4 4 5 6 5 7	20 22 21 22 4 5 21 5 3	24 23 25 6 9 10 2 7 8
5	16 24 33 17 25 34 18 26 35	41 14 30 43 16 31 47 19 35	10 27 16 11 29 18 12 31 20	3 6 9 6 7 11 9 8 13	24 6 2 23 9 7 25 10 8	4 11 15 11 17 13 15 13 21

Two-dimensional,  $3 \times 2$  process grid:

B,D	0 2 4	1 3 5
0 3	P <sub>00</sub>	P <sub>01</sub>
1 4	P <sub>10</sub>	P <sub>11</sub>
2 5	P <sub>20</sub>	P <sub>21</sub>

The symmetric matrix is distributed block-cyclically in lower storage mode over a  $3 \times 2$  process grid:

p,q	0	1
0	1 * * * * * * *	* * * * * * *
	2 10 * * * * * *	* * * * * * *
	3 11 20 * * * * *	* * * * * * *
	10 18 27 4 15 4 * * *	17 36 15 3 * * * *
	11 19 28 5 17 6 * * *	19 49 16 6 1 * * *
	12 20 29 6 19 8 * * *	23 64 20 9 2 1 * * *
1	4 12 21 * * * * *	2 * * * * * *
	5 13 22 * * * * *	3 1 * * * * *
	6 14 23 * * * * *	5 4 5 * * * *
	13 21 30 7 21 10 20 * *	29 81 21 2 3 3 * * *
	14 22 31 8 23 12 22 4 *	31 10 25 4 4 5 * * *
	15 23 32 9 25 14 21 5 3	37 12 26 6 5 7 * * *
2	7 15 24 1 * * * *	7 9 6 * * * *
	8 16 25 2 11 * * *	11 16 10 * * * *
	9 17 26 3 13 2 * *	13 25 11 * * * *
	16 24 33 10 27 16 24 6 2	41 14 30 3 6 9 4 * *
	17 25 34 11 29 18 23 9 7	43 16 31 6 7 11 11 17 *
	18 26 35 12 31 20 25 10 8	47 19 35 9 8 13 15 13 21

where \* means you do not have to store a value in that position in the local array. However, these storage positions are required.

Notice that the local arrays are not symmetric.

Array descriptor DESC\_A contains the following:

DESC_A()	Symbolic name	Value
1	DTYPE_A	1
2	CTXT_A	BLACS context
3	M_A	18
4	N_A	18
5	MB_A	3
6	NB_A	3
7	RSRC_A	0
8	CSRC_A	0
9	LLD_A	6

For more information on how to store symmetric matrices, see the *ESSL Version 3 Release 1.1 Guide and Reference* manual.

**Distributing an Upper Triangular Matrix:** This example shows how the data for a global upper triangular matrix  $A$  with block size of  $2 \times 2$  is distributed block-cyclically over a  $2 \times 3$  process grid. Assume the following  $12 \times 12$  global upper triangular matrix  $A$  with 36 blocks:

B,D	0	1	2	3	4	5
0	2 1	2 13	13 10	15 21	26 31	7 5
	0 3	4 4	11 23	41 45	59 67	1 8
1	0 0	5 9	6 9	33 65	21 14	9 4
	0 0	0 7	16 8	7 33	3 7	5 3

2	0 0	0 0	11 25	10 5	23 7	10 6
	0 0	0 0	0 13	36 12	3 13	5 6
3	0 0	0 0	0 0	17 49	14 1	7 2
	0 0	0 0	0 0	0 19	64 16	1 7
4	0 0	0 0	0 0	0 0	23 81	6 15
	0 0	0 0	0 0	0 0	0 29	9 4
5	0 0	0 0	0 0	0 0	0 0	5 3
	0 0	0 0	0 0	0 0	0 0	0 4

Two-dimensional,  $2 \times 3$  process grid:

B,D	0 3	1 4	2 5
0	P <sub>00</sub>	P <sub>01</sub>	P <sub>02</sub>
2			
4			
1	P <sub>10</sub>	P <sub>11</sub>	P <sub>12</sub>
3			
5			

The following local arrays are distributed block-cyclically in upper-triangular storage mode over a  $2 \times 3$  process grid:

p,q	0	1	2
0	2 1 15 21	2 13 26 31	13 10 7 5
	* 3 41 45	4 4 59 67	11 23 1 8
	* * 10 5	* * 23 7	11 25 10 6
	* * 36 12	* * 3 13	* 13 5 6
	* * * *	* * 23 81	* * 6 15
	* * * *	* * * 29	* * 9 4
	* * 33 65	5 9 21 14	6 9 9 4
1	* * 7 33	* 7 3 7	16 8 5 3
	* * 17 49	* * 14 1	* * 7 2
	* * * 19	* * 64 16	* * 1 7
	* * * *	* * * *	* * 5 3
	* * * *	* * * *	* * * 4

where “\*” means you do not have to store a value in that position in the local array. However, these storage positions are required.

Notice the local arrays are not upper triangular.

Array descriptor DESC\_A contains the following:

DESC_A()	Symbolic name	Value
1	DTYPE_A	1
2	CTXT_A	BLACS context
3	M_A	12
4	N_A	12
5	MB_A	2
6	NB_A	2
7	RSRC_A	0
8	CSRC_A	0
9	LLD_A	6

For more information on how to store triangular matrices, see the *ESSL Version 3 Release 1.1 Guide and Reference* manual.

## Specifying Sparse Matrices for the Fortran 90 and Fortran 77 Sparse Linear Algebraic Equations

For the Fortran 90 and Fortran 77 sparse linear algebraic equation subroutines, you must use the sparse utility subroutines provided with Parallel ESSL to build the sparse matrices on each process in the process grid. This sections shows the calling sequence arguments associated with the sparse matrix *A*.

### Fortran 90 Sparse Linear Algebraic Equation Subroutines

This section contains the following sections:

- “Calling Sequence Arguments for the Sparse Matrix”
- “Derived Data Types” on page 59

**Calling Sequence Arguments for the Sparse Matrix:** This section describes the calling sequence arguments associated with a sparse matrix *A*.

*Table 28. Calling Sequence Arguments for the Sparse Matrix*

Arguments	Meaning
<i>a</i>	is the local part of the sparse matrix <i>A</i> and specified as derived data type D_SPMAT. For more details about D_SPMAT, see “Derived Data Type D_SPMAT” on page 59.
<i>ia</i>	is the row index of the sparse matrix <i>A</i> .
<i>ja</i>	is the column index of the sparse matrix <i>A</i> .
<i>desc_a</i>	is the array descriptor for the sparse matrix <i>A</i> and specified as derived data type DESC_TYPE. For more details about DESC_TYPE, see “Derived Data Type DESC_TYPE” on page 59.
<i>parts</i>	is a user-supplied subroutine that specifies a mapping between a global index for an element in the global sparse matrix and its corresponding storage location on one or more processes.  For details about how you must define the PARTS subroutine, see “Programming Considerations for the Parts Subroutine (Fortran 90 and Fortran 77)” on page 62.

**Derived Data Types:** Some of the arguments of the Fortran 90 sparse linear algebraic equations and their utility subroutines are derived data types.

For more information on derived data types, see the XL Fortran manuals.

*Derived Data Type D\_SPMAT:* Table 29 describes the components of D\_SPMAT that you must provide as input to the PSPINS subroutine. In addition to the components you provide, Parallel ESSL creates other components as necessary that are only for internal use.

*Table 29. Components of D\_SPMAT*

Components of D_SPMAT	Description	Scope
M	Number of local rows	<b>Local</b>
N	Number of local columns	<b>Local</b>
FIDA	Storage mode for the submatrix	Global
AS	Pointer to the submatrix, which contains the coefficients.	<b>Local</b>
IA1	Pointer to the column numbers of each non-zero element in the submatrix.	<b>Local</b>
IA2	Pointer to the starting positions of each row of the submatrix and one position past the end of the submatrix.	<b>Local</b>
<b>Note:</b> The AS, IA1, and IA2 components, which are described in this table depend on how you specify the FIDA component. This description assumes you are using storage by rows. For details about how these components must be specified and their special restrictions, see the appropriate argument descriptions in “PSPINS — Inserts Local Data into a General Sparse Matrix” on page 613.		

*Derived Data Type DESC\_TYPE:* Parallel ESSL builds the array descriptor, *desc\_a*, which is specified as derived data type DESC\_TYPE, and its components, as follows:

- PADALL allocates space for the array descriptor and initializes its components.
- PSPINS updates some components of the array descriptor.
- PSPASB makes final updates to some components of the array descriptor.

MATRIX\_DATA is one component of the array descriptor. Table 30 describes the elements of DESC\_A%MATRIX\_DATA that you may want to reference. However, your application programs should not modify the components of the array descriptor directly. These components should only be updated with calls to PSPINS and PSPASB.

*Table 30. Elements of DESC\_A%MATRIX\_DATA(\_)*

MATRIX_DATA(_)	Name	Description	Data Type	Limits	Scope
1	DEC_TYPE	Type of data distribution	Fullword integer	Internal format	Global

Table 30. Elements of DESC\_A%MATRIX\_DATA(\_) (continued)

MATRIX_DATA(_)	Name	Description	Data Type	Limits	Scope
2	CTXT	BLACS context	Fullword integer	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M	Number of rows in the global general sparse matrix <i>A</i>	Fullword integer	$M \geq 0$ and $M = N$	Global
4	N	Number of columns in the global general sparse matrix <i>A</i>	Fullword integer	$N \geq 0$ and $M = N$	Global
5	N_ROW	Number of local rows	Fullword integer	$N\_ROW \geq 1$	Local
6	N_COL	Number of local columns†	Fullword integer	$N\_COL \geq 1$	Local
†DESC_A%MATRIX_DATA(6) is stable after you have placed a call to PSPASB.					

## Fortran 77 Sparse Linear Algebraic Equation Subroutines

This section contains the following sections:

- “Calling Sequence Arguments for the Sparse Matrix”
- “Array Descriptor” on page 61

**Calling Sequence Arguments for the Sparse Matrix:** This section describes the calling sequence arguments associated with a general sparse matrix *A*.

Table 31. Calling Sequence Arguments for the Sparse Matrix

Arguments	Meaning
<i>as</i>	is the local part of a matrix
<i>ia</i>	is the row index of the sparse matrix.
<i>ja</i>	is the column index of the sparse matrix.
<i>ia1</i>	is the local part of an array containing the sparse matrix indices.
<i>ia2</i>	is the local part of an array containing the sparse matrix indices.
<i>infoa</i>	is an integer array for a matrix. For details about <i>infoa</i> see Table 32.
<i>desc_a</i>	is an array descriptor for the sparse matrix. For details about <i>desc_a</i> see “Array Descriptor” on page 61.
<i>parts</i>	is a user-supplied subroutine that specifies a mapping between a global index for an element in the global sparse matrix and its corresponding storage location on one or more processes.  For details about how you must define the PARTS subroutine, see “Programming Considerations for the Parts Subroutine (Fortran 90 and Fortran 77)” on page 62.

Table 32. Elements of INFOA()

INFOA()	Description	Scope
1	Length of an array for a matrix	Local
2	Length of an array containing sparse matrix indices.	Local

Table 32. Elements of INFOA() (continued)

INFOA()	Description	Scope
3	Length of an array containing sparse matrix indices.	Local
4	Storage format of the matrix.	Global
5	Type of matrix.	Global
6	Number of local rows.	Local
7	Number of local columns.	Local
8 through 30	Reserved for internal use.	—
If <i>infoa</i> is in a subroutine calling sequence, you must always specify a value for INFOA(1), INFOA(2), and INFOA(3).		

**Array Descriptor:** An integer array descriptor, *desc\_a*, is needed to establish a mapping between the global general sparse matrix *A* and its corresponding distributed memory location. You must specify an array descriptor length, DLEN, in DESC\_A(11) on input to PADINIT:

- For the maximum length you should need, use the following formulas to calculate the length of the array descriptor, DLEN.  
If there is no overlap:  

$$DLEN = 33 + 3(np) + n + (N\_COL) + (np - 1)(N\_ROW) + (N\_COL - N\_ROW)$$
 If there is no overlap,  $33 + 3(np) + 4n$  is an upper bound for DLEN.  
 If overlap occurs, add at most to DLEN:  

$$3(np) + 1 + 2(np)(N\_ROW)$$
 where:
  - $N\_ROW \leq n$
  - $N\_COL \leq n$
  - $N\_ROW$  is approximately  $n/np$
  - $n$  is the order of the global general sparse matrix *A*.
  - $np$  is the number of processes in the process grid.
- Use the following formula(s) to calculate a more typical value of the length of the array descriptor, DLEN:  

$$33 + 3(np) + \alpha n \leq DLEN \leq 33 + 6(np) + 3n$$
 where:
  - $1 < \alpha \leq 2$
  - $n$  is the order of the global general sparse matrix *A*.
  - $np$  is the number of processes in the process grid.

**Note:** The actual length of the array descriptor depends on the sparse matrix structure and therefore is known after a call to PDSPASB.

Parallel ESSL builds the remaining elements in the array descriptor, as follows:

- PADINIT initializes the array descriptor.
- PDSPINS updates parts of the array descriptor.
- PDSPASB makes final updates to some parts of the array descriptor.

You may want to use some of the values in *desc\_a* to build vector *b* containing the right-hand side and vector *x* containing initial guess to the solution. (Parallel ESSL

creates other elements in the array descriptor that are for internal use only.) Table 33 describes the elements of the array descriptor that you may want to reference. Your application programs should not modify the elements of the array descriptor directly. The elements should only be updated with calls to PDSPINS and PDSPASB.

Table 33. Elements of DESC\_A()

DESC_A()	Name	Description	Data Type	Limits	Scope
1	DEC_TYPE	Type of data distribution	Fullword integer	Internal format	Global
2	CTXT	BLACS context	Fullword integer	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M	Number of rows in the global general sparse matrix <i>A</i>	Fullword integer	$M \geq 0$ and $M = N$	Global
4	N	Number of columns in the global general sparse matrix <i>A</i>	Fullword integer	$N \geq 0$ and $M = N$	Global
5	N_ROW	Number of local rows <sup>†</sup>	Fullword integer	$1 \leq N\_ROW \leq n$	Local
6	N_COL	Number of local columns <sup>‡</sup>	Fullword integer	$1 \leq N\_COL \leq n$	Local
11	DLEN	Length of the array descriptor	Fullword integer	See the formulas shown in the beginning of this section.	Global

<sup>†</sup>DESC\_A(5) is stable after you have placed a call to PADINIT. You can use this value to calculate *lprcs* in PDSPGPR.

<sup>‡</sup>DESC\_A(6) is stable after you have placed a call to PDSPASB. You can use this value to calculate *lprcs* in PDSPGPR.

DESC\_A(7) through DESC\_A(10) are only for internal use.

DESC\_A(12) through DESC\_A(DLEN) are only for internal use.

## Programming Considerations for the Parts Subroutine (Fortran 90 and Fortran 77)

This section describes how to design and code the *parts* subroutine for use by the Parallel ESSL Fortran 90 and Fortran 77 sparse linear algebraic equation subroutines and their utility subroutines.

You must supply a separate subroutine that is callable by Parallel ESSL. You must specify the name of the subroutine in the *parts* argument. This subroutine name is selected by you. You must declare *parts* as an external subroutine in your application program.

**Coding and Designing the Parts Subroutine for the Sparse Subroutines:** The *parts* subroutine specifies the mapping between a global index for an element in the global general sparse matrix *A* and its corresponding storage location on a process or processes (if overlap occurs).



You should design the *parts* subroutine so it receives, as input, *global\_index*, *n*, and *np*. It also must return to Parallel ESSL, as output, the information in the *pv* and *nv* arguments indicating the storage location of *global\_index* on one or more processes.

*Syntax:*

<b>Fortran</b>	CALL PARTS ( <i>global_index</i> , <i>n</i> , <i>np</i> , <i>pv</i> , <i>nv</i> )
<b>C</b>	parts (& <i>global_index</i> , & <i>n</i> , & <i>np</i> , <i>pv</i> , & <i>nv</i> );
<b>C++</b>	extern "Fortran" void parts(const int &, const int &, const int &, int *, const int &);  parts ( <i>global_index</i> , <i>n</i> , <i>np</i> , <i>pv</i> , <i>nv</i> );

*On Entry:*

*global\_index*

is an input scalar argument containing an integer that indicates the global index for an element in the global general sparse matrix *A*, where:  
 $1 \leq \textit{global\_index} \leq n$ .

*n* is an input scalar argument containing an integer that indicates the order of the global general sparse matrix *A*, where:  $n \geq 0$ .

*np* is an input scalar argument containing an integer that indicates the number of processes in the process grid, where:  $np > 0$ .

*On Output:*

*pv* is an output array containing integers that identify which processes are receiving the global index, *global\_index*, where:  $0 \leq \textit{pv}(i) < np$  and  $1 \leq i \leq nv$ .

*nv* is an output scalar argument containing an integer that indicates the number of unique processes specified in the *pv* argument, where:  
 $1 \leq nv \leq np$ .

*Notes:*

1. The *parts* subroutine can be coded in Fortran, C, or C++. However, for C and C++ programs, all the arguments must be passed by reference.

**Examples for the PARTS Subroutine:** Examples of how you could code *parts* for different types of data distribution are shown in:

- “PART\_BLOCK (Block Data Distribution)” on page 956
- “Block Data Distribution for a C Program”
- “PARTBCYC (Block-Cyclic Data Distribution)” on page 956
- “PARTRAND (Random Data Distribution)” on page 957

*Block Data Distribution for a C Program:*

```
void part_block(global_indx,n,nnodes,pv,nv)
int *global_indx,*n,*nnodes,*pv,*nv;
{
    int dim_block;
    dim_block = (*n + *nnodes - 1)/(*nnodes);
    *nv = 1;
    pv[*nv-1] = (*global_indx - 1)/dim_block;
}
```

## Specifying Sequences for the Fourier Transforms

This section shows how to use block-column distribution to distribute a two- or three-dimensional sequence over a one-dimensional process grid. It also describes how some of the two- and three-dimensional complex sequences are stored in FFT-packed storage mode.

### Two-Dimensional Sequence

Following is a two-dimensional sequence using zero-based indexing where the first dimension is  $n1$ , and the second dimension is  $n2$ :

$$\begin{array}{ccccccc} a_{0,0} & a_{0,1} & \cdot & \cdot & \cdot & a_{0,n2-1} \\ a_{1,0} & a_{1,1} & \cdot & \cdot & \cdot & a_{1,n2-1} \\ \cdot & \cdot & & & & \cdot \\ \cdot & \cdot & & & & \cdot \\ \cdot & \cdot & & & & \cdot \\ a_{n1-1,0} & a_{n1-1,1} & \cdot & \cdot & \cdot & a_{n1-1,n2-1} \end{array}$$

**Distributing Data:** For the Fourier transform subroutines, a two-dimensional sequence is distributed over a one-dimensional process grid, using block-column distribution. The process grid must be arranged as a row ( $1 \times q$ , where  $q$  is the number of processes).

**Note:** Two-dimensional sequences can be thought of as two-dimensional matrices. The term sequence is used because it is traditional for Fourier transforms.

You must distribute the input sequence sequentially to the processes in the process grid, using block-column distribution. Parallel ESSL also returns the output sequence using block-column distribution. The output sequence may be returned in normal or transposed form.

A sequence can be distributed unevenly; that is, one process in the process grid can receive an array that is smaller than other processes. It can also happen that some processes receive no data. “Example 2” on page 805 shows an example of uneven data distribution.

$\text{LOCq}(n)$  represents the number of columns that a process would receive if  $n$  is distributed block over  $q$  processes. You need to calculate  $\text{LOCq}(n)$  for each process, as follows:

- The number of columns,  $\text{LOCq}(n)$ , that processes  $P_{0,0}$  through  $P_{0,k-1}$  receive is calculated as follows:  

$$\text{LOCq}(n) = \text{NB2} = (n+q-1)/q$$
- The number of columns,  $\text{LOCq}(n)$ , that process  $P_{0,k}$  receives is calculated as follows:  

$$\text{LOCq}(n) = n - (q-1)(\text{NB2})$$
- Processes  $P_{0,k+1}$  through  $P_{0,q-1}$  would not receive any data. This may happen if there is not enough data to distribute to all the specified processes.

where:

- $n$  represents the following:
  - $n$  is the second dimension,  $n2$ , of the sequence (for normal form)

- $n$  is the first dimension,  $n1$ , of the sequence (for transposed form and the sequence is not stored in FFT-packed storage mode)
- $n$  is  $n1/2$  (for transposed form and the sequence is stored in FFT-packed storage mode)
- $q$  is the number processes in the process grid
- $P_{0,k}$  is the process that receives the last block of data. For uneven data distribution,  $P_{0,k}$  would receive an array that is smaller than the other processes receive.

Following is an example of block-column distribution for a two-dimensional sequence over a one-dimensional, row-oriented process grid.

Global sequence of size  $8 \times 12$ :

B,D	0	1	2	3
0	0 10 20	30 40 50	60 70 80	90 100 110
	1 11 21	31 41 51	61 71 81	91 101 111
	2 12 22	32 42 52	62 72 82	92 102 112
	3 13 23	33 43 53	63 73 83	93 103 113
	4 14 24	34 44 54	64 74 84	94 104 114
	5 15 25	35 45 55	65 75 85	95 105 115
	6 16 26	36 46 56	66 76 86	96 106 116
	7 17 27	37 47 57	67 77 87	97 107 117

Row-oriented,  $1 \times 4$  process grid:

B,D	0	1	2	3
0	$P_{00}$	$P_{01}$	$P_{02}$	$P_{03}$

Local arrays:

p,q	0	1	2	3
0	0 10 20	30 40 50	60 70 80	90 100 110
	1 11 21	31 41 51	61 71 81	91 101 111
	2 12 22	32 42 52	62 72 82	92 102 112
	3 13 23	33 43 53	63 73 83	93 103 113
	4 14 24	34 44 54	64 74 84	94 104 114
	5 15 25	35 45 55	65 75 85	95 105 115
	6 16 26	36 46 56	66 76 86	96 106 116
	7 17 27	37 47 57	67 77 87	97 107 117

An example of the distribution of a two-dimensional sequence in a Fortran 90 program is shown in Appendix B, “Sample Programs.” See the following:

- The subroutine `initialize-scale` in “Module Scale” on page 919, which determines the parameters to be used for block distribution, ultimately setting up the correct parameters for distributing an FFT sequence.
- The subroutine `get-diffusion_matrix` in “Module Fourier” on page 911, which shows how a local array can be assigned values.
- The subroutine `rlocal_to_rglobal` in “Module Scale” on page 919, which shows gathering the local portions of the block-distributed real array to generate the corresponding global sequence/matrix.

**FFT-Packed Storage Mode:** The output sequence for `PSRCFT2` and `PDCRCFT2`, and the input sequence for `PSRCFT2` and `PDCRCFT2` are stored in FFT-packed storage mode because they consist of complex-conjugate, even symmetric data.

For FFT-packed storage mode, only certain elements of the complex-conjugate, even symmetric data are stored. This section describes how the complex elements of sequence  $y$ , which is the output sequence for PSRCFT2 and PDRCFT2, and the input sequence for PSCRFT2 and PDCRFT2, are stored in global matrices  $Y$  and  $X$ , respectively.

For example, suppose  $y$  is the two-dimensional sequence to be stored in FFT-packed storage mode for PDRCFT2. The following list describes how the elements in  $y$  correspond to the elements in the global matrix  $Y$ :

- The real part of  $y_{0,0}$  is stored in the real part of  $Y_{0,0}$
- The real part of  $y_{0,n2/2}$  is stored in the imaginary part of  $Y_{0,0}$
- The real part of  $y_{n1/2,0}$  is stored in the real part of  $Y_{n2/2,0}$
- The real part of  $y_{n1/2,n2/2}$  is stored in the imaginary part of  $Y_{n2/2,0}$
- The elements  $y_{0,1:n2/2-1}$  are stored in elements  $Y_{1:n2/2-1,0}$
- The elements  $y_{n1/2,1:n2/2-1}$  are stored in elements  $Y_{n2/2+1:n2-1,0}$
- The rows  $y_{1:n1/2-1,i}$  are stored in columns  $Y_{i,1:n1/2-1}$

where:

- $n1$  is the first dimension of array  $y$
- $n2$  is the second dimension of array  $y$
- $i = 0, \dots, n2-1$

The remaining elements of  $y$  are not stored because they are the complex conjugates of elements already stored. These relationships are shown in the following equations:

- $y_{0,n2-j} = \bar{y}_{0,j}$  where  $j = 1, \dots, n2/2-1$
- $y_{n1/2,n2-j} = \bar{y}_{n1/2,j}$  where  $j = 1, \dots, n2/2-1$
- $y_{n1-i,0} = \bar{y}_{i,0}$  where  $i = 1, \dots, n1/2-1$
- $y_{n1-i,n2/2} = \bar{y}_{i,n2/2}$  where  $i = 1, \dots, n1/2-1$
- $y_{n1-i,n2-j} = \bar{y}_{i,j}$  where  $i = 1, \dots, n1/2-1$   
and  $j = 1, \dots, n2/2-1, n2/2+1, \dots, n2-1$

where:

- $n1$  is the first dimension of array  $y$
- $n2$  is the second dimension of array  $y$

The following example, which uses zero-based indexing, has complex conjugate, even symmetry. The dimensions of array  $y$  are  $8 \times 8$  (that is  $n1 = n2 = 8$ ), where array  $y$  is:

$$\begin{bmatrix} (111,0) & (-3,23) & (-8,10) & (-9,4) & (-9,0) & (-9,-4) & (-8,-10) & (-3,-23) \\ (10,-10) & (4,4) & (9,3) & (-6,2) & (-1,2) & (-2,1) & (-3,1) & (-5,-3) \\ (6,-4) & (1,3) & (0,2) & (-7,1) & (-1,9) & (-1,4) & (-2,-4) & (-2,-2) \\ (6,-2) & (6,2) & (-5,1) & (-8,8) & (-1,4) & (-1,-1) & (-1,-8) & (-1,-2) \\ (6,0) & (-3,2) & (-9,1) & (-1,5) & (-1,0) & (-1,-5) & (-9,-1) & (-3,-2) \\ (6,2) & (-1,2) & (-1,8) & (-1,1) & (-1,-4) & (-8,-8) & (-5,-1) & (6,-2) \\ (6,4) & (-2,2) & (-2,4) & (-1,-4) & (-1,-9) & (-7,-1) & (0,-2) & (1,-3) \\ (10,10) & (-5,3) & (-3,-1) & (-2,-1) & (-1,-2) & (-6,-2) & (9,-3) & (4,-4) \end{bmatrix}$$

Because zero-based indexing is used,  $y_{0,0} = (111,0)$ ,  $y_{3,2} = (-5,1)$ , and  $y_{5,7} = (6,-2)$ .

In this example, the real part of  $y_{0,0}$  is 111, the real part of  $y_{0,4}$  is -9, the real part of  $y_{4,0}$  is 6, the real part of  $y_{4,4}$  is -1, and their imaginary parts are all zero. For the FFT-packed storage mode, the imaginary parts at these particular positions are not stored. Therefore, the number stored at position  $Y_{0,0}$  is (111,-9), which represents the contents of both  $y_{0,0}$  and  $y_{0,4}$ . The number stored at position  $Y_{4,0}$  is (6,-1), which represents the contents of both  $y_{4,0}$  and  $y_{4,4}$ .

The elements  $y_{0,1:3}$  are stored in  $Y_{1:3,0}$ . The elements  $y_{4,1:3}$  are stored in  $Y_{5:7,0}$ . The rows  $y_{1:3,0:7}$  are stored in columns  $Y_{0:7,1:3}$ . For FFT-packed storage mode, the elements in positions  $y_{0,5:7}$ ,  $y_{4,5:7}$ , and rows  $y_{5:7,0:7}$  are not stored.

Following is the global matrix  $Y$  in FFT-packed storage mode:

B,D	0	1
0	$\begin{bmatrix} (111,-9) & (10,-10) \\ (-3,23) & (4, 4) \\ (-8,10) & (9, 3) \\ (-9, 4) & (-6, 2) \\ (6,-1) & (-1, 2) \\ (-3, 2) & (-2, 1) \\ (-9, 1) & (-3, 1) \\ (-1, 5) & (-5, -3) \end{bmatrix}$	$\begin{bmatrix} (6,-4) & (6,-2) \\ (1, 3) & (6, 2) \\ (0, 2) & (-5, 1) \\ (-7, 1) & (-8, 8) \\ (-1, 9) & (-1, 4) \\ (-1, 4) & (-1,-1) \\ (-2,-4) & (-1,-8) \\ (-2,-2) & (-1,-2) \end{bmatrix}$

Following is a  $1 \times 2$  process grid:

B,D	0	1
0	P <sub>00</sub>	P <sub>01</sub>

After the data has been distributed over the process grid, the following local arrays for  $Y$  are stored in FFT-packed storage mode:

p,q	0	1
0	$\begin{bmatrix} (111,-9) & (10,-10) \\ (-3,23) & (4, 4) \\ (-8,10) & (9, 3) \\ (-9, 4) & (-6, 2) \\ (6,-1) & (-1, 2) \\ (-3, 2) & (-2, 1) \\ (-9, 1) & (-3, 1) \\ (-1, 5) & (-5, -3) \end{bmatrix}$	$\begin{bmatrix} (6,-4) & (6,-2) \\ (1, 3) & (6, 2) \\ (0, 2) & (-5, 1) \\ (-7, 1) & (-8, 8) \\ (-1, 9) & (-1, 4) \\ (-1, 4) & (-1,-1) \\ (-2,-4) & (-1,-8) \\ (-2,-2) & (-1,-2) \end{bmatrix}$

**Example:** The following example shows how to pack data from a two-dimensional array  $X$  into a global array  $XG$ , whose columns could then be block-column distributed among  $q$  processes. Array  $X$  must contain complex-conjugate even symmetric data.

Each of the  $q$  processes would get  $LOCq(n)$  consecutive columns of array  $XG$ . Array  $X$  is stored as  $n1$  rows by  $n2$  columns. Array  $XG$  is stored as  $n2$  rows by  $n1/2$  columns. This is the transposed form required by PSCRFT2 and PDCRFT2 for the input array.

```

PROGRAM PACK2D
IMPLICIT NONE
INTEGER*4 N1,N2,INDEX,JINDEX
PARAMETER(N1 = 64, N2 = 32)
COMPLEX*16 XG(0:N2-1,0:N1/2-1)
COMPLEX*16 X(0:N1-1,0:N2-1)
XG(0,0) = ( REAL(X(0,0)) , REAL(X(0,N2/2)) )
XG(N2/2,0) = ( REAL(X(N1/2,0)) , REAL(X(N1/2,N2/2)) )
DO INDEX = 1 , N2/2-1

```

```

      XG(INDEX,0) = X(0,INDEX)
      XG(N2/2+INDEX,0) = X(N1/2,INDEX)
ENDDO
DO JINDEX = 0,N2-1
DO INDEX = 1,N1/2-1
      XG(JINDEX,INDEX) = X(INDEX,JINDEX)
ENDDO
ENDDO
STOP
END

```

### Three-Dimensional Sequences

Following is a three-dimensional sequence using zero-based indexing where the first dimension is  $n1$ , the second dimension is  $n2$ , and the third dimension is  $n3$ :

#### Plane 0:

$$\begin{array}{cccc}
 a_{0,0,0} & \cdot & \cdot & \cdot & a_{0,n2-1,0} \\
 a_{1,0,0} & \cdot & \cdot & \cdot & a_{1,n2-1,0} \\
 \cdot & & & & \cdot \\
 \cdot & & & & \cdot \\
 \cdot & & & & \cdot \\
 a_{n1-1,0,0} & \cdot & \cdot & \cdot & a_{n1-1,n2-1,0}
 \end{array}$$

#### Plane 1:

$$\begin{array}{cccc}
 a_{0,0,1} & \cdot & \cdot & \cdot & a_{0,n2-1,1} \\
 a_{1,0,1} & \cdot & \cdot & \cdot & a_{1,n2-1,1} \\
 \cdot & & & & \cdot \\
 \cdot & & & & \cdot \\
 \cdot & & & & \cdot \\
 a_{n1-1,0,1} & \cdot & \cdot & \cdot & a_{n1-1,n2-1,1} \\
 & & & & \cdot \\
 & & & & \cdot \\
 & & & & \cdot
 \end{array}$$

#### Plane ( $n3-1$ ):

$$\begin{array}{cccc}
 a_{0,0,n3-1} & \cdot & \cdot & \cdot & a_{0,n2-1,n3-1} \\
 a_{1,0,n3-1} & \cdot & \cdot & \cdot & a_{1,n2-1,n3-1} \\
 \cdot & & & & \cdot \\
 \cdot & & & & \cdot \\
 \cdot & & & & \cdot \\
 a_{n1-1,0,n3-1} & \cdot & \cdot & \cdot & a_{n1-1,n2-1,n3-1}
 \end{array}$$

**Distributing Data:** For the Fourier transform subroutines, a three-dimensional sequence is distributed over a one-dimensional process grid, using block-plane distribution. The process grid must be arranged as a row ( $1 \times q$ , where  $q$  is the number of processes).

**Note:** Three-dimensional sequences can be thought of as three-dimensional matrices. The term sequence is used because it is traditional for Fourier transforms.

You must distribute the three-dimensional input sequence sequentially to the processes in the process grid, using block-plane distribution. Parallel ESSL also returns the output sequence using block-plane distribution. The output sequence may be returned in normal or transposed form.

A sequence can be distributed unevenly; that is, one process in the process grid can receive an array that is smaller than other processes. It can also happen that some processes receive no data. “Example 2” on page 822 shows an example of when a process does not receive any data.

$\text{LOCq}(n)$  represents the number of planes that a process would receive if  $n$  is distributed block over  $q$  processes. You need to calculate  $\text{LOCq}(n)$  for each process, as follows:

- The number of planes,  $\text{LOCq}(n)$ , that processes  $P_{00}$  through  $P_{0,k-1}$  receive is calculated as follows:  

$$\text{LOCq}(n) = \text{NB3} = (n+q-1)/q$$
- The number of planes,  $\text{LOCq}(n)$ , that process  $P_{0,k}$  receives is calculated as follows:  

$$\text{LOCq}(n) = n-(q-1)(\text{NB3})$$
- Processes  $P_{0,k+1}$  through  $P_{0,q-1}$  would not receive any data. This may happen if there is not enough data to distribute to all the specified processes.

where:

- $n$  represents the following:
  - $n$  is the third dimension,  $n3$ , of the sequence (for normal form)
  - $n$  is the first dimension,  $n1$ , of the sequence (for transposed form and the sequence is not stored in FFT-packed storage mode)
  - $n$  is  $n1/2$  (for transposed form and the sequence is stored in FFT-packed storage mode)
- $q$  is the number processes in the process grid
- $P_{0,k}$  is the process that receives the last block of data. For uneven data distribution,  $P_{0,k}$  would receive an array that is smaller than the other processes receive.

Following is an example of block plane distribution for a three-dimensional sequence over a one-dimensional process grid.

Three-dimensional, global sequence with four planes that are of size  $2 \times 2$ :

	Plane 0:	Plane 1:
B,D	0	
0	$\begin{bmatrix} 0 & 1 \\ 10 & 11 \end{bmatrix}$	$\begin{bmatrix} 10 & 101 \\ 11 & 111 \end{bmatrix}$

	L		J
	Plane 2:	Plane 3:	
B,D	1		
0	$\left[ \begin{array}{cc cc} 20 & 21 & 30 & 31 \\ 23 & 24 & 33 & 34 \end{array} \right]$		

Row-oriented,  $1 \times 2$  process grid:

B,D	0	1
0	P <sub>00</sub>	P <sub>01</sub>

Local arrays:

p,q	0	1
0	$\begin{array}{cc cc} 0 & 1 & 10 & 101 \\ 10 & 11 & 11 & 111 \end{array}$	$\begin{array}{cc cc} 20 & 21 & 30 & 31 \\ 23 & 24 & 33 & 34 \end{array}$

**FFT-Packed Storage Mode:** The output sequence for PSRCFT3 and PDRCFT3, and the input sequence for PSRCFT3 and PDRCFT3 are stored in FFT-packed storage mode because they consist of complex-conjugate, even symmetric data.

For FFT-packed storage mode, only certain elements of the complex-conjugate, even symmetric data are stored. This section describes how the complex elements of sequence  $y$ , which is the output sequence for PSRCFT3 and PDRCFT3, and the input sequence for PSRCFT3 and PDRCFT3, are stored in global matrices  $Y$  and  $X$ , respectively.

For example, suppose  $y$  is the three-dimensional sequence to be stored in FFT-packed storage mode for PDRCFT3. The following list describes how the elements in  $y$  correspond to the complex elements in the global matrix  $Y$ :

- The real part of  $y_{0,0,0}$  is stored the real part of  $Y_{0,0,0}$
- The real part of  $y_{0,0,n3/2}$  is stored in the imaginary part of  $Y_{0,0,0}$
- The elements  $y_{0,0,1:n3/2-1}$  are stored in elements  $Y_{1:n3/2-1,0,0}$
- The real part of  $y_{0,n2/2,0}$  is stored in the real part of  $Y_{n3/2,0,0}$
- The real part of  $y_{0,n2/2,n3/2}$  is stored in the imaginary part of  $Y_{n3/2,0,0}$
- The elements  $y_{0,n2/2,1:n3/2-1}$  are stored in elements  $Y_{n3/2+1:n3-1,0,0}$
- The real part of  $y_{n1/2,0,0}$  is stored in the real part of  $Y_{0,n2/2,0}$
- The real part of  $y_{n1/2,0,n3/2}$  is stored in the imaginary part of  $Y_{0,n2/2,0}$
- The elements  $y_{n1/2,0,1:n3/2-1}$  are stored in elements  $Y_{1:n3/2-1,n2/2,0}$
- The real part of  $y_{n1/2,n2/2,0}$  is stored in the real part of  $Y_{n3/2,n2/2,0}$
- The real part of  $y_{n1/2,n2/2,n3/2}$  is the imaginary part of  $Y_{n3/2,n2/2,0}$
- The elements  $y_{n1/2,n2/2,1:n3/2-1}$  are stored in elements  $Y_{n3/2+1:n3-1,n2/2,0}$
- The rows  $y_{0,1:n2/2-1,i}$  are stored in columns  $Y_{i,1:n2/2-1,0}$
- The rows  $y_{n1/2,1:n2/2-1,j}$  are stored in columns  $Y_{j,n2/2+1:n2-1,0}$
- The planes  $y_{1:n1/2-1,i,j}$  are stored in planes  $Y_{j,i,1:n1/2-1}$

where:

- $i = 0, \dots, n2-1$
- $j = 0, \dots, n3-1$
- $n1$  is the first dimension of array  $y$
- $n2$  is the second dimension of array  $y$
- $n3$  is the third dimension of array  $y$



The remaining elements of  $y$  are not stored because they are the complex conjugates of elements already stored. These relationships are shown in the following equations:

- $y_{0,0,n3-k} = \bar{y}_{0,0,k}$  where  $k = 1, \dots, n3/2 - 1$
- $y_{0,n2/2,n3-k} = \bar{y}_{0,n2/2,k}$  where  $k = 1, \dots, n3/2 - 1$
- $y_{n1/2,0,n3-k} = \bar{y}_{n1/2,0,k}$  where  $k = 1, \dots, n3/2 - 1$
- $y_{n1/2,n2/2,n3-k} = \bar{y}_{n1/2,n2/2,k}$  where  $k = 1, \dots, n3/2 - 1$
- $y_{0,n2-j,0} = \bar{y}_{0,j,0}$  where  $j = 1, \dots, n2/2 - 1$
- $y_{0,n2-j,n3/2} = \bar{y}_{0,j,n3/2}$  where  $j = 1, \dots, n2/2 - 1$
- $y_{n1/2,n2-j,0} = \bar{y}_{n1/2,j,0}$  where  $j = 1, \dots, n2/2 - 1$
- $y_{n1/2,n2-j,n3/2} = \bar{y}_{n1/2,j,n3/2}$  where  $j = 1, \dots, n2/2 - 1$
- $y_{n1-i,0,0} = \bar{y}_{i,0,0}$  where  $i = 1, \dots, n1/2 - 1$
- $y_{n1-i,0,n3/2} = \bar{y}_{i,0,n3/2}$  where  $i = 1, \dots, n1/2 - 1$

- $y_{n1-i,n2/2,0} = \bar{y}_{i,n2/2,0}$  where  $i=1, \dots, n1/2-1$
- $y_{n1-i,n2/2,n3/2} = \bar{y}_{i,n2/2,n3/2}$  where  $i=1, \dots, n1/2-1$
- $y_{0,n2-j,n3-k} = \bar{y}_{0,j,k}$  where  $j=1, \dots, n2/2-1$  and  $k=1, \dots, n3/2-1, n3/2+1, \dots, n3-1$
- $y_{n1/2,n2-j,n3-k} = \bar{y}_{n1/2,j,k}$  where  $j=1, \dots, n2/2-1$  and  $k=1, \dots, n3/2-1, n3/2+1, \dots, n3-1$
- $y_{n1-i,0,n3-k} = \bar{y}_{i,0,k}$  where  $i=1, \dots, n1/2-1$  and  $k=1, \dots, n3/2-1, n3/2+1, \dots, n3-1$
- $y_{n1-i,n2/2,n3-k} = \bar{y}_{i,n2/2,k}$  where  $i=1, \dots, n1/2-1$  and  $k=1, \dots, n3/2-1, n3/2+1, \dots, n3-1$
- $y_{n1-i,n2-j,0} = \bar{y}_{i,j,0}$  where  $i=1, \dots, n1/2-1$  and  $j=1, \dots, n2/2-1, n2/2+1, \dots, n2-1$
- $y_{n1-i,n2-j,n3/2} = \bar{y}_{i,j,n3/2}$  where  $i=1, \dots, n1/2-1$  and  $j=1, \dots, n2/2-1, n2/2+1, \dots, n2-1$
- $y_{n1-i,n2-1-j,n3-1-k} = \bar{y}_{i,j,k}$  where  $i=1, \dots, n1/2-1$  and  $j=1, \dots, n2/2-1, n2/2+1, \dots, n2-1$  and  $k=1, \dots, n3/2-1, n3/2+1, \dots, n3-1$

where:

- $n1$  is the first dimension of array  $y$
- $n2$  is the second dimension of array  $y$
- $n3$  is the third dimension of array  $y$

The following example, which uses zero-based indexing, has complex-conjugate, even symmetry. The dimensions of array  $y$  are  $4 \times 4 \times 4$  (that is  $n1 = n2 = n3 = 4$ ).

**Plane 0:**

$$y_{0:3,0:3,0} = \begin{bmatrix} (30,0) & (2,-3) & (-0.3,0) & (2,3) \\ (-1,0.7) & (-1,-4) & (-2,-0.7) & (0.5,-2) \\ (-2,0) & (-2,-0.6) & (2,0) & (-2,0.6) \\ (-1,-0.7) & (0.5,2) & (-2,0.7) & (-1,4) \end{bmatrix}$$

**Plane 1:**

$$y_{0:3,0:3,1} = \begin{bmatrix} (2,-2) & (-1,1) & (0.7,-2) & (-3,-2) \\ (2,2) & (-2,-1) & (-0.5,3) & (0.04,0.5) \end{bmatrix}$$

$$\begin{bmatrix} (-0.4, 3) & (-0.009, -3) & (0.9, 0.1) & (-1, -0.2) \\ (-2, -2) & (-2, -1) & (-0.5, 2) & (0.1, 0.005) \end{bmatrix}$$

**Plane 2:**

$$y_{0:3,0:3,2} = \begin{bmatrix} (3, 0) & (0.3, 0.5) & (0.1, 0) & (0.3, -0.5) \\ (-0.3, -2) & (1, -3) & (2, 3) & (-7, 3) \\ (2, 0) & (2, -1) & (1, 0) & (2, 1) \\ (-0.3, 2) & (-0.7, -3) & (2, -3) & (1, 3) \end{bmatrix}$$

**Plane 3:**

$$y_{0:3,0:3,3} = \begin{bmatrix} (2, 2) & (-3, 2) & (0.7, 2) & (-1, -1) \\ (-2, 2) & (1, -0.005) & (-0.5, -2) & (-0.2, 1) \\ (-0.4, -3) & (-1, 0.2) & (0.9, -0.1) & (-0.009, 3) \\ (2, -2) & (0.04, -0.5) & (-0.5, -3) & (-2, 1) \end{bmatrix}$$

Because zero-based indexing is used,  $y_{0,0,0} = (30, 0)$ ,  $y_{2,1,1} = (-0.009, -3)$ , and  $y_{3,1,3} = (0.04, -0.5)$ .

In this example, the real part of  $y_{0,0,0}$  is 30, the real part of  $y_{0,0,2}$  is 3, and their imaginary parts are zero. For the FFT-packed storage mode, the imaginary parts at these particular positions are not stored. Therefore, the element stored at position  $Y_{0,0,0}$  is (30,3), which represents the contents of both  $y_{0,0,0}$  and  $y_{0,0,2}$ .

The element  $y_{0,0,1}$  is stored in the global matrix  $Y_{1,0,0}$  position.

The real part of  $y_{0,2,0}$  is -0.3, the real part of  $y_{0,2,2}$  is 0.1, and their imaginary parts are zero. For the FFT-packed storage mode, the imaginary parts at these particular positions are not stored. Therefore, the element stored at position  $Y_{2,0,0}$  is (-0.3,0.1), which represents the contents of both  $y_{0,2,0}$  and  $y_{0,2,2}$ .

The element  $y_{0,2,1}$  is stored in the global matrix  $Y_{3,0,0}$  position.

The real part of  $y_{2,0,0}$  is -2, the real part of  $y_{2,0,2}$  is 2, and their imaginary parts are zero. For the FFT-packed storage mode, the imaginary parts at these particular positions are not stored. Therefore, the element stored at position  $Y_{0,2,0}$  is (-2,2), which represents the contents of both  $y_{2,0,0}$  and  $y_{2,0,2}$ .

The element  $y_{2,0,1}$  is stored in the global matrix  $Y_{1,2,0}$  position.

The real part of  $y_{2,2,0}$  is 2, the real part of  $y_{2,2,2}$  is 1, and their imaginary parts are zero. For the FFT-packed storage mode, the imaginary parts at these particular positions are not stored. Therefore, the element stored at position  $Y_{2,2,0}$  is (2,1), which represents the contents of both  $y_{2,2,0}$  and  $y_{2,2,2}$ .

The element  $y_{2,2,1}$  is stored in the global matrix  $Y_{3,2,0}$  position.

The rows  $y_{0,1,0:3}$  are stored in columns  $Y_{0:3,1,0}$ . The rows  $y_{2,1,0:3}$  are stored in columns  $Y_{0:3,3,0}$ . The plane  $y_{1,0:3,0:3}$  is stored in plane  $Y_{0:3,0:3,1}$ . For FFT-packed storage mode, the remaining elements do not need to be stored due to symmetry.

Following is the global matrix  $Y$  in FFT-packed storage mode:

**Plane 0:**

$$\begin{array}{c} \text{B,D} \end{array} \quad \begin{array}{c} 0 \end{array} \quad \begin{array}{c} 0 \\ \left[ \begin{array}{cccc} (30, 3) & (2, -3) & (-2, 2) & (-2, -0.6) \\ (2, -2) & (-1, 1) & (-0.4, 3) & (-0.009, -3) \\ (-0.3, 0.1) & (0.3, 0.5) & (2, 1) & (2, 1) \\ (0.7, -2) & (-3, 2) & (0.9, 0.1) & (-1, 0.2) \end{array} \right] \end{array}$$

**Plane 1:**

$$\begin{array}{c} \text{B,D} \end{array} \quad \begin{array}{c} 1 \end{array} \quad \begin{array}{c} 0 \\ \left[ \begin{array}{cccc} (-1, 0.7) & (-1, -4) & (-2, -0.7) & (0.5, -2) \\ (2, 2) & (-2, -1) & (-0.5, 3) & (0.04, 0.5) \\ (-0.3, -2) & (1, -3) & (2, 3) & (-0.7, 3) \\ (-2, 2) & (-0.1, -0.005) & (-0.5, -2) & (-0.2, 1) \end{array} \right] \end{array}$$

Following is a  $1 \times 2$  process grid:

$$\begin{array}{c} \text{B,D} \end{array} \quad \begin{array}{c|c} 0 & 1 \\ \hline 0 & P_{00} \quad P_{01} \end{array}$$

After the data has been distributed over the process grid, the following local arrays for  $Y$  are stored in FFT-packed storage mode:

p,q	0	1
0	$\begin{array}{cccc} (30, 3) & (2, -3) & (-2, 2) & (-2, -0.6) \\ (2, -2) & (-1, 1) & (-0.4, 3) & (-0.009, -3) \\ (-0.3, 0.1) & (0.3, 0.5) & (2, 1) & (2, 1) \\ (0.7, -2) & (-3, 2) & (0.9, 0.1) & (-1, 0.2) \end{array}$	$\begin{array}{cccc} (-1, 0.7) & (-1, -4) & (-2, -0.7) & (0.5, -2) \\ (2, 2) & (-2, -1) & (-0.5, 3) & (0.04, 0.5) \\ (-0.3, -2) & (1, -3) & (2, 3) & (-0.7, 3) \\ (-2, 2) & (-0.1, -0.005) & (-0.5, -2) & (-0.2, 1) \end{array}$

**Example:** The following example shows how to pack data from a three-dimensional array  $X$  into a global array  $XG$ , whose planes could then be block distributed among  $q$  processes. Array  $X$  must contain complex-conjugate even symmetric data.

Each of the  $q$  processes would get  $LOCq(n)$  consecutive planes of array  $XG$ . Array  $X$  is stored as  $n1$  rows by  $n2$  columns by  $n3$  planes. Array  $XG$  is stored as  $n3$  rows by  $n2$  columns by  $n1/2$  planes. This is the transposed form required by PSCRFT3 and PDCRFT3 for the input array.

```

PROGRAM PACK3D
IMPLICIT NONE
INTEGER*4 N1,N2,N3
INTEGER*4 IINDEX,JINDEX,KINDEX
PARAMETER(N1 = 64, N2 = 32, N3 = 48)
COMPLEX*16 XG(0:N3-1,0:N2-1,0:N1/2-1)
COMPLEX*16 X(0:N1-1,0:N2-1,0:N3-1)
XG(0,0,0) = ( REAL(X(0,0,0)) , REAL(X(0,0,N3/2)) )
XG(N3/2,0,0) = ( REAL(X(0,N2/2,0)) , REAL(X(0,N2/2,N3/2)) )
XG(0,N2/2,0) = ( REAL(X(N1/2,0,0)) , REAL(X(N1/2,0,N3/2)) )
XG(N3/2,N2/2,0) = ( REAL(X(N1/2,N2/2,0)) , REAL(X(N1/2,N2/2,N3/2)) )
DO IINDEX = 1 , N3/2-1
  XG(IINDEX,0,0) = X(0,0,IINDEX)
  XG(N3/2+IINDEX,0,0) = X(0,N2/2,IINDEX)
  XG(IINDEX,N2/2,0) = X(N1/2,0,IINDEX)
  XG(N3/2+IINDEX,N2/2,0) = X(N1/2,N2/2,IINDEX)
ENDDO

```

```

DO KINDEX = 0,N3-1
DO JINDEX = 1,N2/2-1
  XG(KINDEX,JINDEX,0) = X(0,JINDEX,KINDEX)
  XG(KINDEX,N2/2+JINDEX,0) = X(N1/2,JINDEX,KINDEX)
ENDDO
ENDDO
DO KINDEX = 0,N3-1
DO JINDEX = 0,N2-1
  DO IINDEX = 1,N1/2-1
    XG(KINDEX,JINDEX,IINDEX) = X(IINDEX,JINDEX,KINDEX)
  ENDDO
ENDDO
ENDDO
STOP
END

```



---

## Chapter 3. Coding and Running Your Program

This chapter explains the Parallel ESSL-specific procedures to follow when coding and running your program.

---

### Coding Tips for Optimizing Parallel Performance

Performance has been the primary objective in the design of the Parallel ESSL subroutines. To achieve this performance goal, the Parallel ESSL subroutines use "state-of-the-art" algorithms tailored to specific operational characteristics of the hardware. In addition, Parallel ESSL will leverage the high performance provided by ESSL for processor computations.

#### Choosing How Many MPI Tasks and Computational Threads to Use

If you are using the Parallel ESSL SMP libraries, you need to specify the number of MPI tasks and the number of computational threads you want ESSL and Parallel ESSL to use. To set the number of computational threads, use the environment variable XLSMPOPTS or OMP\_NUM\_THREADS, as shown in the table below.

Table 34. Specifying the Number of MPI Tasks and the Number of Computational Threads to Use on AIX

MPI Tasks	Computational Threads	Specify one of the following environment variables:		
		XLSMPOPTS	—or—	OMP_NUM_THREADS
1 per processor node	Number of CPUs in the processor node	It is not necessary to set an environment variable because the default is equal to the number of CPUs on the processor node.		
1 per CPU	1	XLSMPOPTS=parthds=1	—or—	OMP_NUM_THREADS=1
Multiple MPI tasks per processor node	$thds = (\text{Number of CPUs in the processor node}) / (\text{Number of MPI tasks per processor node})$	XLSMPOPTS=parthds= $thds$	—or—	OMP_NUM_THREADS= $thds$

If you are using the Parallel ESSL GM libraries, you can only run single-threaded applications.

For further details, see the XLF or C manuals.

### Parallel ESSL Techniques

The following techniques are used by most subroutines to optimize performance:

- Minimizing the impact of communications by exchanging larger blocks of data
- Blocking data to match the processor cache size

The following items also impact performance. They generally depend on the specific parallel routine being called. See the subroutine description in the reference section for any exceptions to these rules.

- Number and types of processors

Choosing the number of processors depends primarily on the problem size. It is reasonable to increase the number of processors, if the global problem size increases sufficiently to keep the amount of local data per process at a

reasonable size. If, however, using more processes, such as 17 rather than 16, causes you to use a one-dimensional grid rather than a two-dimensional grid, performance may be degraded. See the next item.

- Shape of process grid

For most subroutines, using a two dimensional (square or as close to square as possible) grid is suggested. For example, if sixteen processors were used, define a 4 by 4 process grid. For exceptions to this rule, see the subroutine descriptions in the reference section.

- Block size(s)

See the following table for suggested block sizes. The optimal block size depends on the underlying node computations, load balancing, communications, system buffering requirements, problem size, and dimension and shape of the process grid. To achieve optimal performance, generally requires experimentation, but the values specified in Table 35 should provide good performance for most cases. For exceptions to these rules, see the subroutine descriptions in the reference section.

*Table 35. Suggested Block Sizes*

Area	Block Sizes
Level 2 PBLAS	24 (All subroutines, except PDTRSV and PZTRSV)  48 (PDTRSV and PZTRSV)
Level 3 PBLAS	100–200 (Real subroutines) 50–100 (Complex subroutines)
Dense Linear Algebraic Equations, except PDGEQRF and PZGEQRF	100–200 (Real subroutines) 50–100 (Complex subroutines)
Eigensystems Analysis and Singular Value Analysis, PDGEQRF, and PZGEQRF	24
Random Number Generation	Data cache size / 2
<b>Note:</b> The data cache size can be obtained by utilizing a code fragment, shown in “PDURNG — Uniform Random Number Generator” on page 839, under Notes and Coding Rules.	

- For the Parallel ESSL SMP libraries:

- If you are using multiple computational threads, your performance may be improved by setting the following environment variable:  
export MALLOCMULTIHEAP=true

**Note:** For details, see the XLF, C, or AIX manuals.

- If you are using multiple message-passing tasks per node, specify MP\_SHARED\_MEMORY=yes to specify the use of shared memory (instead of IP or a switch) for message passing between tasks running on the same node.
- You should be able to improve performance of production-level code by using the PESSL\_ERROR\_SYNC environment variable to disable error synchronization. For details, see “PESSL\_ERROR\_SYNC Environment Variable” on page 103.



---

## Avoiding Conflicts with Parallel ESSL and ESSL Routine Names

Do not use names for your own subroutines, functions, and global variables that are the same as any of the ESSL or Parallel ESSL routine names. All internal ESSL and Parallel ESSL routine names that are exported begin with the “ESV” prefix, so you should avoid using this prefix for your own routine names.

---

## Coding Your Program

This section contains Parallel ESSL-specific application program coding requirements and considerations for programs coded in Fortran, C, and C++. You have the option of using either the BLACS subroutines as documented below or the C interface for the BLACS as shown in Appendix A, “BLACS Quick Reference Guide,” on page 891. To make a Parallel ESSL call in a parallel application program:

1. Call the BLACS initialization subroutines (BLACS\_GET followed by a call to either BLACS\_GRIDINIT or BLACS\_GRIDMAP), to initialize the process grid. For details on how to do this, see “Using the BLACS” on page 80.
2. Ensure your data has been distributed across your process grid, according to the particular input distribution specified by the Parallel ESSL subroutine. For details on how to do this, see Chapter 2, “Distributing Your Data,” on page 21.
3. Call the Parallel ESSL subroutine on all processes in the process grid (defined earlier through the BLACS initialization calls). The Parallel ESSL subroutine call interfaces are documented in Part 2 of this book.
4. When the Parallel ESSL subroutine returns control to the application program, you process the solution data, which is distributed in accordance with the output distribution specified by the Parallel ESSL subroutine.

To look at an application program outline, see the following:

- “Application Program Outline” on page 87 (For all subroutines, except the sparse linear algebraic equation subroutines.)
- “Application Program Outline for the Fortran 90 Sparse Linear Algebraic Equations and Their Utilities” on page 89
- “Application Program Outline for the Fortran 77 Sparse Linear Algebraic Equations and Their Utilities” on page 90

For an example of the use of Parallel ESSL in a sample Fortran 90 application program solving a thermal diffusion problem, see Appendix B, “Sample Programs,” on page 895.

Parallel ESSL supports both 32-bit- and 64-bit-environment applications. The data model for the 64-bit environment is referred to as LP64. This data model supports 32-bit integers and 64-bit pointers. In accordance with the LP64 data model, all Parallel ESSL integer arguments remain 32 bit.

The *ESSL Guide and Reference* manual contains additional information about coding ESSL subroutine calls in Fortran, C, and C++ programs. That information also applies to Parallel ESSL and is not repeated in this book. The specific topics you may want to reference, that apply to Parallel ESSL, are:

- Coding the calling sequences
- Passing arguments
- Setting up scalar data
- Setting up complex data in C and C++ programs
- Setting up arrays

## Using the BLACS

A parallel machine with  $k$  processes is often thought of as a one-dimensional linear array of processes labeled 0, 1, ...,  $k-1$ . For performance reasons, it is sometimes useful to map this one-dimensional array into a logical two-dimensional rectangular grid, which is also referred to as process grid, of processes. The process grid can have  $p$  process rows and  $q$  process columns, where  $p \times q = k$ . A process can now be indexed by row and column,  $(i,j)$ , where  $0 \leq i < p$  and  $0 \leq j < q$ .

Before calling the Parallel ESSL subroutines, you need to call `BLACS_GET`, followed by a call to either `BLACS_GRIDINIT` or `BLACS_GRIDMAP` to define the size and dimensions of your process grid. This identifies what processes are involved in the communication. You can reinitialize the BLACS, as needed, at various points in your application program to redefine the process grid.

When you initialize the BLACS, you must specify the (total) size  $k$  of the grid to be less than or equal to the number of MPI tasks you are using. If argument values are not valid, an error message is issued and the program is terminated.

An example of initializing the BLACS in a Fortran 90 program is shown in Appendix B, "Sample Programs," on page 895. See the subroutine `initialize_scale` in "Module Scale" on page 919.

### BLACS\_PINFO

You call the `BLACS_PINFO` routine when you want to determine how many processes are available. You can use this information as input into other BLACS routines that set up your process grid.

#### Syntax:

Language	Call Statement
Fortran	<code>CALL BLACS_PINFO (mypnum, nprocs)</code>
C	<code>blacs_pinfo (&amp;mypnum, &amp;nprocs);</code>
C++	<code>extern "FORTRAN" void blacs_pinfo(const int &amp;, const int &amp;); blacs_pinfo (mypnum, nprocs);</code>

#### On Return:

*mypnum*

is the local process rank (for example, message-passing task number) that your program is currently running on.

Returned as: a fullword integer value, where:  $0 \leq \text{mypnum} \leq (\text{nprocs} - 1)$ .

*nprocs*

is the number of processes available for the BLACS to use.

Returned as: a fullword integer value.

### BLACS\_GET

You call the `BLACS_GET` routine when you want the values the BLACS are using for internal defaults. The most common use is in retrieving a default system context for input into `BLACS_GRIDINIT` or `BLACS_GRIDMAP`.

#### Syntax:

Language	Call Statement
Fortran	CALL BLACS_GET ( <i>icontxt</i> , <i>what</i> , <i>val</i> )
C	blacs_get (& <i>icontxt</i> , & <i>what</i> , & <i>val</i> );
C++	extern "FORTRAN" void blacs_get(const int &, const int &, const int &); blacs_get ( <i>icontxt</i> , <i>what</i> , <i>val</i> );

#### On Entry:

*icontxt* has the following meaning:

If *what* = 0 or 2, *icontxt* is ignored.

If *what* = 10, *icontxt* is the integer handle indicating the BLACS context.

Specified as: a fullword integer value.

*what* indicates the BLACS internal to be returned in *val*. For a description of the values of *what*, see Table 36.

Table 36. Input and Output for BLACS\_GET

Value of <i>what</i>	BLACS Internals That are Returned in <i>val</i>
0	Handle indicating the default system context
2	BLACS debug level
10	Handle indicating the system context used to define the BLACS context whose handle is <i>icontxt</i> .  You can redefine the shape of your process grid by calling BLACS_GET with <i>what</i> =10. For examples on how to do this, see the "Notes" section in "BLACS_GRIDINIT" or "BLACS_GRIDMAP" on page 83.

Specified as: a fullword integer value 0, 2, 10.

#### On Return:

*val* is the value of the BLACS internal, as defined for each value of *what* in Table 36.

Returned as: a fullword integer value.

### BLACS\_GRIDINIT

You call the BLACS\_GRIDINIT routine when you want to map the processes sequentially in row-major order or column-major order into the process grid. You must specify the same input argument values in the calls to BLACS\_GRIDINIT on every process.

#### Syntax:

Language	Call Statement
Fortran	CALL BLACS_GRIDINIT ( <i>icontxt</i> , <i>order</i> , <i>nprow</i> , <i>npcol</i> )
C	blacs_gridinit (& <i>icontxt</i> , & <i>order</i> , & <i>nprow</i> , & <i>npcol</i> , <i>lorder</i> );
C++	extern "FORTRAN" void blacs_gridinit(const int &, char *, const int &, const int &, size_t); blacs_gridinit ( <i>icontxt</i> , <i>order</i> , <i>nprow</i> , <i>npcol</i> , <i>lorder</i> );

**On Entry:**

*icontxt* is the system context to be used in creating the BLACS context. For examples on obtaining a default system context and reshaping your process grid, see the “Notes” section.

Specified as: a fullword integer value.

*order* indicates how to map processes into the process grid, where:

If *order* = 'R', row-major natural ordering is used. This is the default.

If *order* = 'C', column-major natural ordering is used.

Specified as: a single character; *order* = 'R' or 'C'.

*nprow* is the number of rows in this process grid.

Specified as: a fullword integer where:  $1 \leq nprow \leq p$ .

*npcol* is the number of columns in this process grid.

Specified as: a fullword integer value where:  $1 \leq npcol \leq q$ .

*lorder* is the string length of *order*.

Specified as: a size\_t value where: *lorder* = strlen(*order*).

**On Return:**

*icontxt* is the integer handle to the BLACS context, which is a mechanism for partitioning communication space. A defining property of a context is that a message in a context cannot be sent or received in another context. The BLACS context includes the definition of a grid, and each processor's coordinates in the grid.

Returned as: a fullword integer value.

**Notes:**

1. You may obtain a default system context by calling BLACS\_GET as follows:  
CALL BLACS\_GET(0, 0, *icontxt*)
2. You can redefine the shape of your process grid by calling BLACS\_GET with *what*=10 and then calling BLACS\_GRIDINIT. The following example shows how to create a  $1 \times 4$  process grid, using the context from a  $2 \times 2$  process grid:

```
*
* Define the 1 x 4 process grid
*
CALL BLACS_GET(0, 0, icontxt)
CALL BLACS_GRIDINIT(icontxt, 'R' 2, 2)
.
.
.
*
* Redefine the shape to a 1 x 4 process grid
*
CALL BLACS_GET(icontxt, 10, newcontxt)
CALL BLACS_GRIDINIT(newcontxt, 'R', 1, 4)
```

3. Suppose you specified a total of fifteen processes in your MP\_PROCS environment variable, referred to as  $t_0$  through  $t_{14}$ . You then call BLACS\_GRIDINIT in your Fortran program, as follows:

```
CALL BLACS_GRIDINIT (icontxt, 'R', 3, 4)
```

The processes would be mapped sequentially in row major order into a 3 by 4 process grid as follows:

Table 37. A 3 by 4 process grid

$P_{p,q}$	0	1	2	3
0	$t_0$	$t_1$	$t_2$	$t_3$
1	$t_4$	$t_5$	$t_6$	$t_7$
2	$t_8$	$t_9$	$t_{10}$	$t_{11}$

**Note:** In this example, the process grid is 3 by 4. You must execute a call to Parallel ESSL on all processes whose process row and column index satisfy  $0 \leq i < 3$  and  $0 \leq j < 4$ , respectively.

## BLACS\_GRIDINFO

You call the BLACS\_GRIDINFO routine to obtain the process row and column index.

### Syntax:

Language	Call Statement
Fortran	CALL BLACS_GRIDINFO ( <i>icontxt</i> , <i>nprow</i> , <i>npcol</i> , <i>myrow</i> , <i>mycol</i> )
C	blacs_gridinfo (& <i>icontxt</i> , & <i>nprow</i> , & <i>npcol</i> , & <i>myrow</i> , & <i>mycol</i> );
C++	extern "FORTRAN" void blacs_gridinfo(const int &, const int &, const int &, const int &, const int &);  blacs_gridinfo ( <i>icontxt</i> , <i>nprow</i> , <i>npcol</i> , <i>myrow</i> , <i>mycol</i> );

### On Entry:

*icontxt* is the integer handle to the BLACS context which is a mechanism for partitioning communication space. A defining property of a context is that a message in a context cannot be sent or received in another context. The BLACS context include the definition of a grid, and each process coordinates in the grid.

Specified as: a fullword integer value, returned by BLACS\_GRIDINIT or BLACS\_GRIDMAP.

### On Return:

*nprow* is the number of rows in this process grid.

Specified as: a fullword integer where:  $1 \leq nprow \leq p$ .

*npcol* is the number of columns in this process grid.

Specified as: a fullword integer value where:  $1 \leq npcol \leq q$ .

*myrow* is the process grid row index.

Returned as: a fullword integer value where:  $0 \leq myrow < p$ .

*mycol* is the process grid column index.

Returned as: a fullword integer value where:  $0 \leq mycol < q$ .

## BLACS\_GRIDMAP

You call the BLACS\_GRIDMAP routine when you want to map the processes in a specific manner into a process grid. You pass in a two-dimensional array

containing the process numbers, which is mapped into your new process grid. You must specify the same input argument values in the calls to BLACS\_GRIDMAP on every process.

#### Syntax:

Language	Call Statement
Fortran	CALL BLACS_GRIDMAP ( <i>icontxt</i> , <i>usermap</i> , <i>ldumap</i> , <i>nprow</i> , <i>npcol</i> )
C	<code>blacs_gridmap (&amp;icontxt, usermap, &amp;ldumap, &amp;nprow, &amp;npcol);</code>
C++	extern "FORTRAN" void blacs_gridmap(const int &, int *, const int &, const int &, const int &);  <code>blacs_gridmap (icontxt, usermap, ldumap, nprow, npcol);</code>

#### On Entry:

*icontxt* is the system context to be used in creating the BLACS context. For examples on obtaining a default system context and reshaping your process grid, see the "Notes" section.

Specified as: a fullword integer value.

*usermap*

specifies the process-to-grid mapping. USERMAP(i,j) contains the number of the process to be mapped to the process grid, location (i,j).

Specified as: a two dimensional integer array of size *ldumap* by *npcol*.

*ldumap*

is the leading dimension of the integer array USERMAP.

Specified as: an integer where:  $ldumap \geq nprow$

*nprow*

is the number of rows in this process grid.

Specified as: a fullword integer where:  $1 \leq nprow \leq p$ .

*npcol*

is the number of columns in this process grid.

Specified as: a fullword integer value where:  $1 \leq npcol \leq q$ .

#### On Return:

*icontxt* is the integer handle to the BLACS context which is a mechanism for partitioning communication space. A defining property of a context is that a message in a context cannot be sent or received in another context. The BLACS context include the definition of a grid, and each process coordinates in the grid.

Returned as: a fullword integer value.

#### Notes:

1. You may obtain a default system context by calling BLACS\_GET as follows:  
`CALL BLACS_GET(0, 0, icontxt)`
2. You can redefine the shape of your process grid by calling BLACS\_GET with *what*=10 and then calling BLACS\_GRIDMAP. The following example shows how to create a  $1 \times 4$  process grid, using the context from a  $2 \times 2$  process grid:  
\*  
\* Define the  $1 \times 4$  process grid  
\*  
`CALL BLACS_GET(0, 0, icontxt)`

```
CALL BLACS_GRIDMAP(icontxt, usermap, 2, 2, 2)
      .
      .
      .
*
* Redefine the shape of your 2 × 2 process grid
* to a 1 × 4 process grid
*
CALL BLACS_GET(icontxt, 10, newcontxt)
CALL BLACS_GRIDMAP(newcontxt, usermap, 2, 1, 4)
```

3. Suppose you specified a total of 15 processes in your MP\_PROCS environment variable, referred to as  $t_0$  through  $t_{14}$ . You then called BLACS\_GRIDMAP in your Fortran program, as follows:

```
CALL BLACS_GRIDMAP (icontxt1,USERMAP,5,3,4)
```

Where array USERMAP1 contained the following integer values:

$$\text{USERMAP1} = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 8 & 9 & 10 & 11 \\ 4 & 5 & 6 & 7 \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}$$

then, the processes would be mapped into a 3 by 4 process grid as follows:

Table 38. 3 by 4 process grid

$P_{p,q}$	0	1	2	3
0	$t_0$	$t_1$	$t_2$	$t_3$
1	$t_8$	$t_9$	$t_{10}$	$t_{11}$
2	$t_4$	$t_5$	$t_6$	$t_7$

BLACS\_GRIDMAP sets *icontxt1*. Use the value of *icontxt1* in any subsequent calls to Parallel ESSL to use this process grid.

While the above process grid is active, another overlapping process grid can be defined. Suppose you then called BLACS\_GRIDMAP in your Fortran program as follows:

```
CALL BLACS_GRIDMAP(icontxt2, USERMAP2, 2, 2, 2)
```

where USERMAP contains the following values:

$$\text{USERMAP2} = \begin{bmatrix} 1 & 2 \\ 10 & 11 \end{bmatrix}$$

Then the processes would be mapped into a 2 by 2 process grid as follows:

Table 39. 2 by 2 process grid

$P_{p,q}$	0	1
0	$t_1$	$t_2$
1	$t_{10}$	$t_{11}$

BLACS\_GRIDMAP will set *icontxt2*. Use the value of *icontxt2* in any subsequent calls to Parallel ESSL to use this process grid.

#### Notes:

- In this example, process  $t_1$  is mapped to  $P_{01}$  in the first grid and to  $P_{00}$  in the second grid.
- Both grids can simultaneously be used in your program.

## BLACS\_GRIDEXIT

You call the BLACS\_GRIDEXIT routine to release a BLACS context.

### Syntax:

Language	Call Statement
Fortran	CALL BLACS_GRIDEXIT ( <i>icontxt</i> )
C	blacs_gridexit (& <i>icontxt</i> );
C++	extern "FORTRAN" void blacs_gridexit(const int &);  blacs_gridexit ( <i>icontxt</i> );

### On Entry:

*icontxt* is the integer handle to the BLACS context indicating the BLACS context to be released.

Specified as: a fullword integer value, returned by BLACS\_GRIDINIT or BLACS\_GRIDMAP.

## BLACS\_EXIT

You call the BLACS\_EXIT routine to release all the BLACS context and the memory allocated by the BLACS.

### Syntax:

Language	Call Statement
Fortran	CALL BLACS_EXIT ( <i>continue</i> )
C	blacs_exit (& <i>continue</i> );
C++	extern "FORTRAN" void blacs_exit(const int &);  blacs_exit ( <i>continue</i> );

### On Entry:

*continue*

has the following meaning:

If *continue* = 0, all the BLACS context and memory allocated by the BLACS are released. In addition, Parallel ESSL calls MPI\_Finalize to exit from MPI. There can only be one call to MPI\_Finalize in your program. Therefore, at the end of your program, you should call BLACS\_EXIT with *continue* = 0 or call MPI\_Finalize directly.

If *continue*  $\neq$  0, the BLACS contexts and memory allocated by the BLACS are released, however, you can continue using MPI. When you are finished using MPI, you need to remember to call MPI\_Finalize directly.

Specified as: a fullword integer.

## Using Extrinsic Procedures—The Fortran 90 Sparse Linear Algebraic Equation Subroutines

In Fortran 90 programs, the Parallel ESSL sparse linear algebraic equation subroutines are invoked with the CALL statement, using the features of Fortran 90—generic interfaces, optional and keyword arguments, assumed-shape arrays, and modules.



The Fortran 90 sparse linear algebraic equation subroutines require that an explicit interface be provided for each extrinsic procedure entry in the scope where it is called, using an interface block. The interface blocks for the Parallel ESSL subroutines are provided for you in the module F90SPARSE, so you do not have to code the interface blocks yourself. In the beginning of your program, before any other specification statements, you must code the statement:

```
use f90sparse
```

This gives the XL Fortran compiler access to the interface blocks. For examples of where to code this statement in your program, see “Application Program Outline for the Fortran 90 Sparse Linear Algebraic Equations and Their Utilities” on page 89.

If you are accessing the Fortran 90 sparse linear algebraic equation subroutines from a 64-bit-environment program, you need to modify your existing Fortran compilation procedures to specify the location of the 64-bit module F90SPARSE. (See “Running Your Program on AIX” on page 91 and “Running Your Program on Linux” on page 94.)

For further details on coding the CALL statement and other related aspects of Fortran 90 programs, see the Fortran manuals.

## Setting Up the Parallel ESSL Header File for C and C++

Before you can call the Parallel ESSL subroutines from your C or C++ program:

- You must have the Parallel ESSL header file, `pessl.h`, installed on your system. This header file allows you to code your function calls as described in Part 2 of this book. You should check with your system support group to verify that the appropriate Parallel ESSL header file is installed.
- You must code the following statement in the beginning of your C or C++ program:

```
#include <pessl.h>
```

## Setting Up the C Interface for the BLACS Header File for C and C++

Before you can call the C interface for the BLACS subroutines from your C or C++ program (as described in Appendix A, “BLACS Quick Reference Guide,” on page 891):

- You must have the `Cblacs.h` header file installed on your system.
- You must code the following statement in the beginning of your C or C++ program:

```
#include <Cblacs.h>
```

## Application Program Outline

For the Level 2 and 3 PBLAS, dense and banded linear algebraic equations, and eigensystem analysis and singular value analysis subroutines, this application program outline shows how you can use the BLACS to define a process grid, set up a Type-1 array descriptor, call a Parallel ESSL subroutine, and exit the BLACS. For a complete example, see Appendix B, “Sample Programs,” on page 895.

```

      .
      .
      .

```

```

*

```

```

* Determine my process number and the total number of available

```

```

* processes
*
      CALL BLACS_PINFO(IAM, NNODES)
*
* Define a process grid that is as close to square as possible
*
      NPROW = INT(SQRT(REAL(NNODES)))
      NPCOL = NNODES/NPROW
*
* Get the default system context
* Define the process grid
* Determine my process row and column index
*
      CALL BLACS_GET(0, 0, ICONTXT)
      CALL BLACS_GRIDINIT(ICONTXT, 'R', NPROW, NPCOL)
      CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)
*
* Only call the Parallel ESSL subroutine if I am in the process grid
*
      IF (MYROW .LT. NPROW .AND. MYCOL .LT. NPCOL) THEN
*
* Setup input arrays, scalars, array descriptors, etc.
*
          .
          .
*
* NUMROC can be used to return the size of local arrays
* For example, here is one way to setup the descriptor vector for A
*
          DESC_A(1) = DTYPE_A
          DESC_A(2) = ICONTXT
          DESC_A(3) = M_A
          DESC_A(4) = N_A
          DESC_A(5) = MB_A
          DESC_A(6) = NB_A
          DESC_A(7) = RSRC_A
          DESC_A(8) = CSRC_A
          DESC_A(9) = MAX (1, NUMROC(DESC_A(3), DESC_A(5), MYROW, DESC_A(7), NPROW))
          .
          .
          .
*
* CALL Parallel ESSL subroutine
*
      CALL PDTRAN(M, N, ALPHA, A, IA, JA, DESC_A, BETA, C, IC, JC, DESC_C)
*
* Process output arrays, scalars etc.
*
          .
          .
          .
*
* When finished with this process grid, release the process grid.
*
      CALL BLACS_GRIDEXIT(ICONTXT)
      .
      ENDIF
      .
      .
      .
*
* At the end of the program, exit from the BLACS and MPI
*
      CALL BLACS_EXIT(0)

```

```

      .
      .
      .
END

```

## Application Program Outline for the Fortran 90 Sparse Linear Algebraic Equations and Their Utilities

The following is an outline for a application program that is calling the Fortran 90 sparse linear algebraic equation subroutines and their utilities. For a more complete example, see “Example—Using the Fortran 90 Sparse Subroutines” on page 636 or “Fortran 90 Sample Sparse Program” on page 932.

```

USE F90SPARSE
      .
      .
      .
!User-defined subroutine
INTERFACE PARTS
  SUBROUTINE PARTS(...)
    INTEGER GLOBAL_INDEX, N, NP
    INTEGER NV
    INTEGER PV(*)
  END SUBROUTINE PARTS
END INTERFACE
      .
      .
      .
!Define the process grid
CALL BLACS_GET (...)
CALL BLACS_GRIDINIT(...)
CALL BLACS_GRIDINFO(...)
      .
      .
      .
!Allocate space for and initialize array descriptor desc_a.
CALL PADALL(...)

!Allocate space and initialize some values
!for sparse matrix A.
CALL PSPALL(...)

!Allocate and build vectors b and x.
CALL PGEALL(...)

!Build the sparse matrix A with multiple calls to PSPINS.
!Each process has to call PSPINS as many times as
!necessary to insert the local rows it owns.
!Update array descriptor desc_a.
do
  CALL PSPINS(...)
enddo

!Build vectors b and x with multiple calls to PGEINS.
!Each process has to call PGEINS as many times as
!necessary to insert the local elements it owns.
do
  CALL PGEINS(...)
enddo

!Finalize the sparse matrix A and array descriptor desc_a
CALL PSPASB(...)

```

```

!Finalize the vectors b and x.
!Matrix A and array descriptor desc_a
!must be finalized before calling PGEASB.
CALL PGEASB(...)

!Prepare preconditioner
CALL PSPGPR(...)

!Call solver
CALL PSPGIS(...)

!Cleanup and exit.
!Deallocate vectors b and x
!Deallocate matrix A and the preconditioner data structure PRC
CALL PGEFREE(...)
CALL PSPFREE(...)

!Deallocate the array descriptor desc_a only after
!vectors b and x, and matrix A are deallocated.
CALL PADFREE(...)

.
.
.
CALL BLACS_GRIDEXIT(...)
CALL BLACS_EXIT(...)

```

## Application Program Outline for the Fortran 77 Sparse Linear Algebraic Equations and Their Utilities

The following is an outline for a application program that is calling the Fortran 77 sparse linear algebraic equation subroutines and their utilities. For a complete example, see “Example—Using the Fortran 77 Sparse Subroutines” on page 669 or “Fortran 77 Sample Sparse Program” on page 941.

```

.
.
.
EXTERNAL PARTS
.
.
.
!Define the process grid
CALL BLACS_GET (...)
CALL BLACS_GRIDINIT(...)
CALL BLACS_GRIDINFO(...)
.
.
.
!Initialize array descriptor desc_a.
CALL PADINIT(...)

!Initialize some values
!for sparse matrix A.
CALL PDSPINIT(...)

!Build the sparse matrix A with multiple calls to PDSPINS.
!Each process has to call PDSPINS as many times as
!necessary to insert the local rows it owns.
!Update array descriptor desc_a.
do
  CALL PDSPINS(...)
enddo

```

```

!Build vectors b and x with multiple calls to PDGEINS.
!Each process has to call PDGEINS as many times as
!necessary to insert the local elements it owns.
do
    CALL PDGEINS(...)
enddo

!Finalize the sparse matrix A and array descriptor desc_a
CALL PDSPASB(...)

!Finalize the vectors b and x.
CALL PDGEASB(...)

!Prepare preconditioner
CALL PDSPGPR(...)

!Call solver
CALL PDSPGIS(...)

.
.
.
CALL BLACS_GRIDEXIT(...)
CALL BLACS_EXIT(...)

```

---

## Running Your Program

This section describes **both the Parallel ESSL-specific and ESSL-specific changes** you need to make to your job procedures for compiling, linking, and running your program.

You can use any procedures you are currently using to compile, link, and run your Fortran, C, and C++ programs, as long as you make the necessary modifications required by Parallel ESSL.

## Running Your Program on AIX

The following notes apply to running your program on AIX.

### Notes:

1. The default search path for the Parallel ESSL and ESSL libraries is: `/usr/lib`. (Note that `/lib` is a symbolic link to `/usr/lib`.)  
If the libraries are installed somewhere else, add the path name of that directory to the beginning of the **LIBPATH** environment variable, being careful to keep `/usr/lib` in the path. The correct **LIBPATH** setting is needed both for linking and executing the program.  
For example, if you installed the Parallel ESSL libraries in `/home/me/lib` you would issue ksh commands similar to the following in order to compile and link a program:  

```
LIBPATH=/home/me/lib:/usr/lib
export LIBPATH
mpxlf -o myprog myprog.f -lessl -lpessl -lblacs
```

After setting the **LIBPATH** command, the `/home/me/lib` directory is the directory that gets searched first for the necessary libraries. This same search criterion is used at both compile and link time and run time.
2. For the Parallel ESSL SMP libraries, you can use the **XLSMPOPTS** or **OMP\_NUM\_THREADS** environment variable to specify options that affect SMP execution. For details, see Table 34 on page 77.
3. If you are accessing Parallel ESSL from a 64-bit-environment program, you must add the **-q64** compiler option.

4. Parallel ESSL supports the XL Fortran compile-time option **-qextname**. For details, see the Fortran manuals.
5. The ESSL and Parallel ESSL libraries are shared libraries and must be used in conjunction with each other. Equivalent subroutines with the same names in other libraries (such as libblas.a) will not be used even if they are specified on the command line in place of the ESSL library.
6. The MPICH-GM scripts are installed in /usr/bin.

## Parallel ESSL for AIX—Dynamic Linking Versus Static Linking

Only dynamic linking is supported for programs using Parallel ESSL for AIX.

## Parallel ESSL for AIX—Fortran Program Procedures

You do not need to modify your existing Fortran compilation procedures when using Parallel ESSL for AIX unless you are accessing the Fortran 90 sparse linear algebraic equation subroutines from a 64-bit-environment program. In that case, you must add:

**-I/usr/lpp/pessl.rte.common/include/64**

to your compilation command (as shown in the tables that follow).

When linking and running your program, you must modify your existing job procedures for Parallel ESSL for AIX, to set up the necessary libraries.

If you are accessing Parallel ESSL for AIX from a Fortran program, you can compile and link using the commands in the following tables.

Table 40. Fortran program compile and link commands for use with SMP libraries

Application Mode	Command
32-bit	mpxlf_r -O xyz.f -lesslsmpl -lpesslsmpl -lblacssmpl
64-bit	mpxlf_r -O -q64 xyz.f -lesslsmpl -lpesslsmpl -lblacssmpl mpxlf_r -O -q64 xyz.f -lesslsmpl -lpesslsmpl -lblacssmpl -I/usr/lpp/pessl.rte.common/include/64

Table 41. Fortran program compile and link commands for use with GM libraries

Application Mode	Command
32-bit	mpif77 -fc=xlfr -O -qnosave xyz.f -lessl -lpesslgm -lblacsgm
64-bit	mpif77 -fc=xlfr -O -qnosave -q64 xyz.f -lessl -lpesslgm -lblacsgm

Parallel ESSL for AIX supports the XL Fortran compile-time option **-qextname**. For details, see the Fortran manuals.

An example of a makefile is shown in “Sample Makefiles and Run Script for AIX” on page 967.

## Parallel ESSL for AIX—C Program Procedures

The Parallel ESSL for AIX header files pessl.h and Cblacs.h, used for C and C++ programs, are installed in the /usr/include directory. You do not need to modify your existing C compilation procedures when using Parallel ESSL for AIX, unless you want to specify your own definitions for complex data.

If you do want to specify your own definitions for short- and long-precision complex data, add **-D\_CMPLX** and **-D\_DCMPLX**, respectively, to your compile

and link command. Otherwise, you automatically use the definitions of short- and long-precision complex data provided in the Parallel ESSL for AIX header file (as shown in the tables that follow).

When linking and running your program, you must modify your existing job procedures for Parallel ESSL for AIX, to set up the necessary libraries.

If you are accessing Parallel ESSL for AIX from a C program, you can compile and link using the commands shown in the following tables.

Table 42. C program compile and link commands for use with SMP libraries

Application Mode	Command
32-bit	<code>mpicc_r -O xyz.c -lesslsmpl -lpesslsmpl -lblacssmpl</code>
	<code>mpicc_r -O -D_CMPLX -D_DCMLX xyz.c -lesslsmpl -lpesslsmpl -lblacssmpl</code>
64-bit	<code>mpicc_r -O -q64 xyz.c -lesslsmpl -lpesslsmpl -lblacssmpl</code>
	<code>mpicc_r -O -D_CMPLX -D_DCMLX -q64 xyz.c -lesslsmpl -lpesslsmpl -lblacssmpl</code>

Table 43. C program compile and link commands for use with GM libraries

Application Mode	Command
32-bit	<code>mpicc -cc=cc_r -O xyz.c -lessl -lpesslgm -lblacsgm</code>
64-bit	<code>mpicc -cc=cc_r -O -q64 xyz.c -lessl -lpesslgm -lblacsgm</code>

## Parallel ESSL for AIX—C++ Program Procedures

The Parallel ESSL for AIX header files `pessl.h` and `cbacsl.h`, used for C and C++ programs, are installed in the `/usr/include` directory. When using Parallel ESSL for AIX, the compiler option `-qnocinc=/usr/include/pessl` must be specified.

If you are using the IBM Open Class<sup>®</sup> Complex Mathematics Library, you automatically use the definition of short-precision complex data provided in the Parallel ESSL for AIX header file. If you prefer to specify your own definition for short-precision complex data, add `-D_CMPLX` to your commands (as shown in the tables that follow). Otherwise, Parallel ESSL for AIX will use the IBM Open Class Complex Mathematics Library or the Standard Numerics Library, as described in *ESSL Guide and Reference*.

If you prefer to explicitly specify that you want to use the Standard Numerics Library facilities for complex arithmetic, add `-D_ESV_COMPLEX_` to your command as shown in the table below.

The Parallel ESSL for AIX header file supports two alternatives for declaring scalar output arguments. By default, the arguments are declared to be type reference. If you prefer for them to be declared as pointers, add `-D_ESVCPTR` to your commands as shown in the table below.

When linking and running your program, you must modify your existing job procedures for Parallel ESSL for AIX to set up the necessary libraries.

If you are accessing Parallel ESSL for AIX from a C++ program, you can compile and link using the commands shown in the following tables.

Table 44. C++ program compile and link commands for use with SMP libraries

Application Mode	Command
32-bit	<code>mpCC_r -O xyz.C -lesslsmpl -lpesslsmpl -lblacssmpl -qnocinc=/usr/include/pessl</code>
	<code>mpCC_r -O -D_CMLX xyz.C -lesslsmpl -lpesslsmpl -lblacssmpl -qnocinc=/usr/include/pessl</code>
	<code>mpCC_r -O -D_ESV_COMPLEX_ xyz.C -lesslsmpl -lpesslsmpl -lblacssmpl -qnocinc=/usr/include/pessl</code>
	<code>mpCC_r -O -D_ESVCPTR xyz.C -lesslsmpl -lpesslsmpl -lblacssmpl -qnocinc=/usr/include/pessl</code>
64-bit	<code>mpCC_r -O -q64 xyz.C -lesslsmpl -lpesslsmpl -lblacssmpl -qnocinc=/usr/include/pessl</code>
	<code>mpCC_r -O -D_CMLX -q64 xyz.C -lesslsmpl -lpesslsmpl -lblacssmpl -qnocinc=/usr/include/pessl</code>
	<code>mpCC_r -O -D_ESV_COMPLEX_ -q64 xyz.C -lesslsmpl -lpesslsmpl -lblacssmpl -qnocinc=/usr/include/pessl</code>
	<code>mpCC_r -O -D_ESVCPTR -q64 xyz.C -lesslsmpl -lpesslsmpl -lblacssmpl -qnocinc=/usr/include/pessl</code>

Table 45. C++ program compile and link commands for use with GM libraries

Application Mode	Command
32-bit	<code>mpicc -cc=xlc_r -O xyz.C -lessl -lpesslgm -lblacsgm</code>
64-bit	<code>mpicc -cc=xlc_r -O -q64 xyz.C -lessl -lpesslgm -lblacsgm</code>

## Running Your Program on Linux

The following notes apply to running your program on Linux.

### Notes:

1. The default search paths for the Parallel ESSL libraries are as follows:

#### 32-bit libraries

The default search path is: **/usr/lib**

#### 64-bit libraries

The default search path is: **/usr/lib64**

If the libraries are installed somewhere else, you need to set the link-time and run-time library search paths. There are two ways to set these search paths:

- Use one of the following compile/link options:

**-R (or -rpath)** Writes the specified run-time library search paths into the executable program.

**-L** Searches the library search paths at link time, but does not write them into the executable as run-time library search paths.

—or—

- Use one of the following environment variables:

**LD\_LIBRARY\_PATH** Specifies the directories that are to be searched for libraries at run time.

**LD\_RUN\_PATH** Specifies the directories that are to be searched for libraries at both link and run time.

For example, if you installed the Parallel ESSL 32-bit libraries in /home/me/lib, you would issue ksh commands similar to the following in order to compile and link a program:



```
LD_LIBRARY_PATH=/home/me/lib:$LD_LIBRARY_PATH
LD_RUN_PATH=/home/me/lib:$LD_RUN_PATH
export LD_LIBRARY_PATH
export LD_RUN_PATH
mpif77 -fc=xlf_r -O -qnosave xyz.f -lessl -lpesslgm -lblacsgm
```

The result would be that the /home/me/lib directory is the directory that gets searched at link time and run time.

2. If you are accessing Parallel ESSL from a 64-bit–environment program, you must add the **-q64** compiler option.
3. Parallel ESSL supports the XL Fortran compile-time option **-qextname**. For details, see the Fortran manuals.
4. To access all MPICH-GM scripts, put the following in your path:
  - *prefix/bin*

where *prefix* is the location where the MPICH-GM binary tree is installed. For more information on link options and environment variables, see the manpage for the **ld** command.

5. The commands in the tables below assume that you installed the IBM compilers in the default directory, /opt/ibmcmp. If you used different directories, you need to make the appropriate changes to the -L, -R, and -I options.

## Parallel ESSL for Linux—Dynamic Linking Versus Static Linking

Only dynamic linking is supported for programs using Parallel ESSL for Linux.

## Parallel ESSL for Linux—Fortran Program Procedures

You do not need to modify your existing Fortran compilation procedures when using Parallel ESSL for Linux unless you are accessing the Fortran 90 sparse linear algebraic equation subroutines from a 64-bit–environment program. In that case, you must add:

**-I/opt/ibmmath/pessl/3.2/include/64**

to your compilation command (as shown in the table below).

When linking and running your program, you must modify your existing MPICH-GM job procedures for Parallel ESSL for Linux, to set up the necessary libraries.

If you are accessing Parallel ESSL for Linux from a Fortran program, you can compile and link using the commands in the table below.

Application Mode	Command
32-bit	mpif77 -fc=xlf_r -O -qnosave xyz.f -lessl -lpesslgm -lblacsgm
64-bit	mpif77 -fc=xlf_r -O -qnosave -q64 xyz.f -lessl -lpesslgm -lblacsgm

Parallel ESSL for Linux supports the XL Fortran compile-time option **-qextname**. For details, see the Fortran manuals.

An example of a makefile is shown in “Sample Makefiles and Run Script for Linux” on page 980.

## Parallel ESSL for Linux—C Program Procedures

The Parallel ESSL for Linux header files `pessl.h` and `Cblacs.h`, used for C and C++ programs, are installed in the `/usr/include` directory.

You do not need to modify your existing C compilation procedures when using Parallel ESSL for Linux, unless you want to specify your own definitions for complex data.

If you do want to specify your own definitions for short- and long-precision complex data, add `-D_CMPLX` and `-D_DCMPLX`, respectively, to your compile and link command. Otherwise, you automatically use the definitions of short- and long-precision complex data provided in the Parallel ESSL for Linux header file (as shown in the table below).

When linking and running your program, you must modify your existing MPICH-GM job procedures for Parallel ESSL for Linux, to set up the necessary libraries.

If you are accessing Parallel ESSL for Linux from a C program, you can compile and link using the commands shown in the table below.

Application Mode	Command
32-bit	<code>mpicc -cc=cc_r -O xyz.c -lessl -lpesslgm -lblacsgm -lxl90_r -lxlomp_ser -lxlmath -lmpichfarg -L/opt/ibmcmp/xlf/9.1/lib -R/opt/ibmcmp/xlf/9.1/lib</code>
64-bit	<code>mpicc -cc=cc_r -O -q64 xyz.c -lessl -lpesslgm -lblacsgm -lxl90_r -lxlomp_ser -lxlmath -lmpichfarg -L/opt/ibmcmp/xlf/9.1/lib64 -R/opt/ibmcmp/xlf/9.1/lib64</code>

## Parallel ESSL for Linux—C++ Program Procedures

The Parallel ESSL for Linux header files `pessl.h` and `Cblacs.h`, used for C and C++ programs, are installed in the `/usr/include` directory.

The Parallel ESSL for Linux header file supports two alternatives for declaring scalar output arguments. By default, the arguments are declared to be type reference. If you prefer for them to be declared as pointers, add `-D_ESVCPTR` to your commands as shown in the table below.

When linking and running your program, you must modify your existing MPICH-GM job procedures for Parallel ESSL for Linux to set up the necessary libraries.

If you are accessing Parallel ESSL for Linux from a C++ program, you can compile and link using the commands shown in the table below.

**Note:** The `mpiCC` (uppercase “CC”) scripts use compiler options not supported by the IBM XL C/C++ Advanced Edition compiler. To avoid loader warning messages, the `mpicc` (lowercase “cc”) scripts are used in the table below.

Application Mode	Command
32-bit	<code>mpicc -cc=xlc_r -O xyz.C -lessl -lpesslgm -lblacsgm -lxl90_r -lxlomp_ser -lxlmath -lmpichfarg -L/opt/ibmcmp/xlf/9.1/lib -R/opt/ibmcmp/xlf/9.1/lib</code>

Application Mode	Command
64-bit	mpicc -cc=xlc_r -O -q64 xyz.C -lessl -lpesslgm -lblacsgm -lxlf90_r -lxlomp_ser -lxlfmath -lmpichfarg -L/opt/ibmcmp/xlf/9.1/lib64 -R/opt/ibmcmp/xlf/9.1/lib64



---

## Chapter 4. Migrating Your Programs

This chapter explains many aspects of migrating your application programs.

---

### Migrating to Parallel ESSL for AIX Version 3 Release 2

No changes to your application programs are required if you are migrating from Parallel ESSL for AIX Version 3 Release 1 to Parallel ESSL for AIX Version 3 Release 2.

---

### Migrating to Parallel ESSL for Linux Version 3 Release 2

No changes to your application programs are required if you are migrating from Parallel ESSL for Linux Version 3 Release 1 Modification 1 to Parallel ESSL for Linux Version 3 Release 2.

Parallel ESSL for Linux Version 3 Release 2 does not support SLES8. In most cases, binary compatibility does not exist between SLES8 and SLES9. Therefore, SLES8 applications must be recompiled and rebuilt on SLES9.

Red Hat 3 is no longer supported.

On Linux, if you are accessing Parallel ESSL from a C or C++ program, you must change your compile and link commands so that they specify the IBM XL Fortran Advanced Edition Version 9.1 for Linux libraries.

---

### Migrating to Parallel ESSL Version 3 Release 1

The Parallel ESSL Serial Libraries are used with the Parallel Environment MPI Signal Handling Library. Parallel Environment 4.1 provides only binary compatibility support for the MPI Signal Handling Library. Existing applications that use the Parallel ESSL Serial Libraries will continue to run, but if you are creating new applications you should use the Parallel ESSL SMP libraries.

No changes to your application programs are required if you are migrating from Parallel ESSL Version 2 Release 3 to Parallel ESSL Version 3 Release 1.

**Note:** If you are migrating from a release earlier than Parallel ESSL Version 2 Release 3, see the documentation for Parallel ESSL Version 2 Release 3 at the following IBM Web site:  
[http://www-1.ibm.com/servers/eserver/pseries/library/sp\\_books/essl.html](http://www-1.ibm.com/servers/eserver/pseries/library/sp_books/essl.html)

---

### Migrating from ScaLAPACK to Parallel ESSL

The following information applies to migrating from ScaLAPACK to Parallel ESSL.

#### Migrating from ScaLAPACK 1.5 to Parallel ESSL

If you are currently using the ScaLAPACK 1.5 offerings from the Oak Ridge National Laboratory, Parallel ESSL uses compatible calling sequences with this version of ScaLAPACK.



---

## Chapter 5. Using Error Handling

This chapter provides the following information for your use in dealing with errors:

- How to obtain IBM support.
- What to do about NLS (National Language Support) problems.
- A description of the different types of errors that can occur in Parallel ESSL. It explains what happens when an error occurs and, in some instances, how you can use error handling to obtain further information.
- All of the Parallel ESSL error messages are categorized into the different error types. There is also a description of the error message format.

---

### Where to Find More Information About Errors

Information about errors and how to handle them can be found in the following places:

- Specific errors associated with each Parallel ESSL subroutine are listed under “Error Conditions” in each subroutine description in Part 2 of this book.
- Diagnostic procedures for errors associated with ESSL are provided in the *ESSL Guide and Reference* manual.

---

### Getting Help from IBM Support

Should you require help from IBM in resolving a Parallel ESSL problem, report it and provide the following information, if available and appropriate.

1. Your customer number
2. The Parallel ESSL program number:

**Parallel ESSL for AIX**                      5765-F84

**Parallel ESSL for Linux**                  5765-G18

This is important information that speeds up the correct routing of your call.

3. The version and release of the operating system that you are running on.

**On AIX**                      Enter the following command:  
**oslevel**

**On Linux**                    Enter the following command:  
**uname -a**

This is important information that speeds up the correct routing of your call.

4. The names and versions of key products being run.

**On AIX**                      Enter the following command:  
**lspp -h product**

where the appropriate values of *product* are listed in Table 46 on page 102.

Table 46. Product File Set Names for Parallel ESSL for AIX

Product File Sets	Descriptive Name
essl.*	ESSL for AIX
pessl.*	Parallel ESSL for AIX
ppe.poe	Parallel Operating Environment
xlfrte	XL Fortran Run-Time Environment
xlsmprte	SMP Run-Time Environment
xlfcmp	XL Fortran Compiler
vac.C	C for AIX Compiler
vacpp.cmp.C	VisualAge® C++ Professional for AIX Compiler

**On Linux** Enter the following command:

- **rpm -q package**

where the appropriate values of *package* are listed in Table 47.

Table 47. Package Names for Parallel ESSL for Linux

Product Package	Descriptive Name
essl.rte	ESSL for Linux
pessl.rte	Parallel ESSL for Linux
xlfrte	XL Fortran Run-Time Environment
xlsmprte	SMP Run-Time Environment
xlfcmp	XL Fortran Compiler
vac.cmp	C for Linux Compiler
vacpp.cmp	VisualAge C++ Professional for Linux Compiler

5. The type of Parallel ESSL library (SMP or GM) being run.
6. The message that is returned when an error is detected.
7. Any error message relating to core dumps.
8. The compiler listings, including compiler options in effect, and any run-time listings produced
9. Program changes made in comparison with a previous successful run
10. A small test case demonstrating the problem using the minimum number of statements and variables, including input data

Consult your IBM Service representative for more assistance.

## National Language Support

For National Language Support (NLS), all Parallel ESSL subroutines display messages located in externalized message catalogs. English versions of the message catalogs are shipped with the product, but your site may be using its own translated message catalogs. The environment variable **NLSPATH** is used by the various Parallel ESSL subroutines to find the appropriate message catalog. **NLSPATH** specifies a list of directories to search for message catalogs. The directories are searched, in the order listed, to locate the message catalog. In



resolving the path to the message catalog, **NLSPATH** is affected by the value of the environment variables **LC\_MESSAGES** and **LANG**.

The Parallel ESSL message catalogs are in English and are located in the following directories:

**On AIX**

```
/usr/lib/nls/msg/C
/usr/lib/nls/msg/En_US
/usr/lib/nls/msg/en_US
```

**On Linux**

```
/usr/share/locale/en_US/LC_MESSAGES
/usr/share/locale/C
```

If your site is using its own translations of the message catalogs, consult your system administrator for the appropriate value of **NLSPATH** or **LANG**. For additional information on NLS and message catalogs, see *IBM AIX General Programming Concepts: Writing and Debugging Programs*.

---

## PESSL\_ERROR\_SYNC Environment Variable

The **PESSL\_ERROR\_SYNC** environment variable allows you to enable and disable error synchronization. If error synchronization is disabled, the first process containing input-argument error(s) to finish computing, issues its error message(s) and terminates Parallel ESSL processing on all processes. Therefore, you should only disable error synchronization when your application program is debugged.

```
PESSL_ERROR_SYNC=no
-or-
PESSL_ERROR_SYNC=NO

export PESSL_ERROR_SYNC
```

This causes Parallel ESSL to disable error synchronization in all calls to the Parallel ESSL subroutines.

If you do not set the environment variable or you set something other than 'no' or 'NO', Parallel ESSL uses error synchronization in all calls to the Parallel ESSL subroutines.

---

## Dealing with Errors

At run time, you can encounter a number of different types of errors that are specifically related to the use of the Parallel ESSL subroutines:

- Program exceptions
- Input-argument errors (001-299, 800-999)
- Computational errors (300-399)
- Resource errors (400-499)
- Communication errors (500-599)
- Miscellaneous errors (700-799)

This section explains what causes these errors, what happens when they occur (all are terminating, except computational errors), and what you can do to fix them.

This section also explains what to do when you receive informational and attention messages (600-699).

## Program Exceptions

The program exceptions you can encounter in Parallel ESSL are described in the *ANSI/IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985*.

## Input-Argument Errors

This section describes how Parallel ESSL implements input-argument error checking when error synchronization is enabled. For more information on the `PESSL_ERROR_SYNC` environment variable, which allows you to enable or disable error synchronization, see “`PESSL_ERROR_SYNC` Environment Variable” on page 103.

Two types of input-argument error checking may be performed:

- First, on each participating process, Parallel ESSL checks the validity of most input-arguments in multiple stages.

When all the input-arguments in one stage are valid, Parallel ESSL checks the validity of the input-arguments in the next stage, and so on. (The number of errors and stages that can occur for each subroutine are listed under its “Error Conditions” section, which is in Part 2 of this book.)

When an input argument is invalid on all participating processes in the parallel environment, a single comprehensive error message is issued, rather than one for each process. (This is indicated in the error message by `Process(-1,-1)`.) Otherwise, an error message is issued from each process where the discrepancy occurred.

For all subroutines except `DESCINIT`, Parallel ESSL then terminates your program on all processes, and any arguments in the stages that follow are not checked. When this occurs, you should use standard programming techniques to diagnose and fix the errors.

For subroutine `DESCINIT`, Parallel ESSL does not terminate your program. Instead, Parallel ESSL issues message 0040-295, sets *info* to the first argument found to have an invalid value, and returns a valid array descriptor.

- Next, Linear Algebraic Equations and Eigensystem Analysis subroutines check to ensure that global scalar arguments are the same on all participating processes.

If the value of the global scalar argument on all processes except  $P_{00}$  does not match the value of the argument at process  $P_{00}$ , a single error message is issued. (This is indicated in the error message by `Process(-1,-1)`.) Otherwise, an error message is issued from each process where the discrepancy occurred. Parallel ESSL then terminates your program on all processes, and you should use standard programming techniques to diagnose and fix the errors.

For all other Parallel ESSL subroutines, the global scalar arguments are not checked to ensure they are the same for all processes.

### How This Differs from ESSL:

The capabilities of `ERRSET`, `ERRSAV`, and `ERRSTR`, supported in ESSL, are not provided in Parallel ESSL.

Using the capabilities of `ERRSET`, `ERRSAV`, and `ERRSTR` with your ESSL subroutines does not affect the Parallel ESSL subroutines.

For the Fourier transform subroutines, an invalid transform length is not recoverable, as in ESSL. Parallel ESSL checks the validity of the transform length you provide to the Fourier transform subroutine. If it is not an acceptable value, a

Parallel ESSL input-argument error message is issued, containing the next larger acceptable transform length required for successful computing of a Fourier transform. See the appropriate subroutine for additional constraints on valid transform lengths. Your program is then terminated on all processes. You should correct the value and rerun your program.

## Computational Errors

Parallel ESSL computational errors are errors that occur in the computational data, such as in your vectors and matrices, during a computation—for example, the detection of a singular system during a factorization. (The computational errors that can occur for each subroutine, are listed under “Computational Errors”.) When a computational error occurs, Parallel ESSL issues an error message containing information key to the diagnosis of the error—such as the location in the input matrix where the singularity occurred. Any subroutine that issues a computational error has an *info* argument in its calling sequence. For all the Parallel ESSL subroutines, *info* is a global argument containing fullword integers, except in the tridiagonal subroutines. For these tridiagonal subroutines, *info* is a local argument containing fullword integers.

When a computational error occurs, your program continues to execute. After each call where a computational error can occur, you should check the *info* output argument to see if an error occurred and take the appropriate action. When a computational error occurs, you should assume that the results are unpredictable. The result of the computation is valid only if no errors have occurred.

### How This Differs from ESSL:

The way you handle computational errors for Parallel ESSL differs from how you handle them for ESSL. This is because the capabilities of ERRSET, ERRSAV, and ERRSTR, supported in ESSL for recoverable computational errors, are **not supported** in Parallel ESSL. This results in the following differences:

- You do not have the option of Parallel ESSL terminating your program when a computational error occurs in an Parallel ESSL subroutine. Control always returns to your program.
- The information about the error is returned to your program through the *info* argument, rather than through a subsequent call to the EINFO subroutine.

Using the capabilities of ERRSET, ERRSAV, and ERRSTR with your ESSL subroutines does not affect the Parallel ESSL subroutines.

## Resource Errors

A resource error occurs when a buffer storage allocation request fails in a Parallel ESSL subroutine. In general, the Parallel ESSL subroutines allocate internal auxiliary storage dynamically as needed. Without sufficient storage, the subroutine cannot complete the computation.

When a buffer storage allocation request fails, a resource error message is issued, and the application program is terminated. You need to reduce the memory constraint on the system or increase the amount of memory available before rerunning the application program.

### Ways to Reduce Memory Constraints:

The following ways may reduce memory constraints:

- If you are using a one-dimensional process grid, change to a two-dimensional process grid, if possible. (Keep the shape of the two-dimensional process grid as close to a square as possible.)
- If you are using a two-dimensional process grid, change the shape of the process grid to be a square or as close to a square as possible.
- Increase the number of nodes. As you increase the number of nodes, keep the process grid as square as possible. For example, if using more processes, such as 17 rather than 16, causes you to use a one-dimensional grid rather than a two-dimensional grid, performance may be degraded.
- Reduce the block sizes.
- Set the leading dimension equal to the number of rows in the local matrix.
- Investigate the load of your process and run in a more dedicated environment.
- Increase your node's paging space.
- Select nodes with more available memory.
- Select nodes that are not being used by other programs.
- **On AIX only:**
  - For a 32-bit environment application, consider specifying the `-bmaxdata` binder option when linking your program. For details see the Fortran publications.
  - Check the setting of your user ID's user limit (`ulimit`). (See the *IBM AIX Commands Reference*).

## Communication Errors

Communication errors are errors that occur when Parallel ESSL encounters problems in communicating between processes—sending and receiving data or synchronizing operations. When a communication error occurs, at least one communication message is issued and the application program is terminated. This is because communication errors usually indicate a serious problem, where it is not feasible to continue.

Be aware that, due to the nature of communication errors, some error messages, including communications error messages from various processes, may be lost.

## Informational and Attention Messages

When you receive an informational or attention message, check your application to determine why the condition was detected. You may decide to change your application so you do not receive the message. For example, if your application called a BLACS routine to send data from one process to the same process, you would receive an attention message.

Parallel ESSL does not terminate your application program, but performance may be degraded.

## Miscellaneous Errors

A miscellaneous error is an error that does not fall under any of the other categories. Miscellaneous errors are checked in stages along with input-argument errors.

If no errors are detected in the first stage, Parallel ESSL checks the next stage, and so on. (The number of errors and stages that can occur for each subroutine are listed under its "Error Conditions" section.)

When Parallel ESSL detects a miscellaneous error, you receive an error message with information on how to correct the problem, your application program is terminated, and any arguments in the stages that follow are not checked.

## ESSL Error Messages

For problems relating directly to ESSL, see the *ESSL Guide and Reference* manual. If the ESSL error resulted from a Parallel ESSL subroutine, see “Getting Help from IBM Support” on page 101 to find out how to report the problem.

## MPI Error Messages

If you receive an MPI error message while calling a BLACS routine, the cause is most likely one of the following:

- The BLACS have not been initialized.
- The context passed to the BLACS routine is not the same as the context obtained from a call to the BLACS\_GET, BLACS\_GRIDINIT, or BLACS\_GRIDMAP routine.

---

## Messages

This section describes the message conventions and lists all messages for input-argument errors, computational errors, resource errors, communication errors, informational and attention messages, and miscellaneous errors.

## Message Conventions

### About Upper- and Lowercase:

The literals, such as ‘N’, ‘T’, ‘U’, and so forth, appear in the messages in this book in uppercase; however, they may be specified in your Parallel ESSL calling sequence in either upper- or lowercase, for example, ‘n’, ‘t’, and ‘u’.

### Message Format:

The Parallel ESSL messages are issued in your output in the following format:

```
rtn-name : 0040-nnn Context(l) Task(k) Process(p,q) Grid P × Q  
message-text
```

Figure 8. Message Format

The parts of the Parallel ESSL message are as follows:

**0040** is the Parallel ESSL component identification number.

*nnn* is the message identification number:

**001–299**

Input-argument error messages

**300–399**

Computational error messages

**400–499**

Resource-allocation error messages

**500–599**

Communications error messages

**600–699**

Informational and attention messages

700–799

Miscellaneous error messages

800–999

Input-argument error messages

**Context**

$l$  is the communication context number defined for this process grid, where  $l$  is an integer. If  $l = -1$ , then the context is invalid; in addition, the process and grid coordinates are set to  $-1$ .

**Task( $k$ )**

is the MPI task identification number.

**Process( $p,q$ )**

are the process grid coordinates, indicating the process where the error occurred.

If  $p = q = -1$  and the context is valid, then the same error occurred on all the processes, but is only reported on  $P_{00}$ .

**Grid  $P \times Q$**

gives the dimensions of the process grid.

*message-text*

describes the nature of the error. The possible unique parts are:

- For the message passing error messages, the argument number of each argument involved in the error is included in the message description as (ARG NO.  $\_$ ).
- Additional information about the error is included in the message. The placement of this information is shown in the messages as ( $\_$ )

## Input-Argument Error Messages (001-299)

0040-001	Context( $\_$ ) Task( $\_$ ) Process( $\_$ , $\_$ ) Grid $\_ \times \_$ The SCOPE (ARG NO. $\_$ ) of a broadcast must be 'R', 'C', or 'A'	0040-006	Context( $\_$ ) Task( $\_$ ) Process( $\_$ , $\_$ ) Grid $\_ \times \_$ The SCOPE is specified by (ARG NO. $\_$ ); therefore, the index of the source process (ARG NO. $\_$ ) must be equal to ( $\_$ ).
0040-002	Context( $\_$ ) Task( $\_$ ) Process( $\_$ , $\_$ ) Grid $\_ \times \_$ UPLO (ARG NO. $\_$ ), which specifies whether an input matrix (ARG NO. $\_$ ) is upper or lower, must be 'U' or 'L'.	0040-007	Context( $\_$ ) Task( $\_$ ) Process( $\_$ , $\_$ ) Grid $\_ \times \_$ The TOPOLOGY parameter (ARG NO. $\_$ ) is invalid.
0040-003	Context( $\_$ ) Task( $\_$ ) Process( $\_$ , $\_$ ) Grid $\_ \times \_$ DIAG (ARG NO. $\_$ ), which specifies whether an input matrix (ARG NO. $\_$ ) is unit, must be 'U' or 'N'.	0040-008	Context( $\_$ ) Task( $\_$ ) Process( $\_$ , $\_$ ) Grid $\_ \times \_$ The requested number of processes ( $\_$ ) is greater than the available number of processes ( $\_$ ).
0040-004	Context( $\_$ ) Task( $\_$ ) Process( $\_$ , $\_$ ) Grid $\_ \times \_$ The process row (ARG NO. $\_$ ) must be greater than or equal to zero and less than the total number of processes in a row.	0040-009	Context( $\_$ ) Task( $\_$ ) Process( $\_$ , $\_$ ) Grid $\_ \times \_$ The requested number of process rows ( $\_$ ) and process columns ( $\_$ ) must be positive.
0040-005	Context( $\_$ ) Task( $\_$ ) Process( $\_$ , $\_$ ) Grid $\_ \times \_$ The process column (ARG NO. $\_$ ) must be greater than or equal to zero and less than the total number of processes in a column.	0040-010	Context( $\_$ ) Task( $\_$ ) Process( $\_$ , $\_$ ) Grid $\_ \times \_$ The number of rows (ARG NO. $\_$ ) in a matrix must be greater than or equal to zero.

0040-011	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The number of columns (ARG NO. <u>  </u> ) in a matrix must be greater than or equal to zero.	0040-024	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The size of the leading dimension (ARG NO. <u>  </u> ) of the local array must be greater than zero.
0040-012	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The block size (ARG NO. <u>  </u> ) must be greater than zero.	0040-025	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The process column (ARG NO. <u>  </u> ) that contains matrix (ARG NO. <u>  </u> ) must be equal to the process column (ARG NO. <u>  </u> ) that contains matrix (ARG NO. <u>  </u> )
0040-014	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The stride (ARG NO. <u>  </u> ) for a vector must be positive.	0040-026	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The process row (ARG NO. <u>  </u> ) that contains matrix (ARG NO. <u>  </u> ) must be equal to the process row (ARG NO. <u>  </u> ) that contains matrix (ARG NO. <u>  </u> )
0040-015	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> (ARG NO. <u>  </u> ) must be a double precision odd whole number greater than or equal to 1.0 and less than 2**48.	0040-027	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The order (ARG NO. <u>  </u> ) of a matrix must be greater than or equal to zero.
0040-016	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> (ARG NO. <u>  </u> ) must be zero or one.	0040-028	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> MTXBLK (ARG NO. <u>  </u> ), which specifies whether an input matrix (ARG NO. <u>  </u> ) is a full block matrix or a single block matrix, must be 'M' or 'B'.
0040-017	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> (ARG NO. <u>  </u> ) must be greater than or equal to zero.	0040-029	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The process row (ARG NO. <u>  </u> ) must be greater than or equal to -1 and less than the total number of rows in the process grid.
0040-018	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> (ARG NO. <u>  </u> ) must be greater than zero.	0040-030	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The process column (ARG NO. <u>  </u> ) must be greater than or equal to -1 and less than the total number of columns in the process grid.
0040-019	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The number of rows (ARG NO. <u>  </u> ) must be less than or equal to the block size (ARG NO. <u>  </u> ).	0040-031	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The argument which specifies whether a matrix (ARG NO. <u>  </u> ) is workspace must be 'Y' or 'N'.
0040-020	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The number of columns (ARG NO. <u>  </u> ) must be less than or equal to the block size (ARG NO. <u>  </u> ).	0040-032	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> TRANS (ARG NO. <u>  </u> ), which specifies the computation to be performed, must be 'N', 'T', or 'C'.
0040-021	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The number of rows (ARG NO. <u>  </u> ) must be less than or equal to the size of the leading dimension (ARG NO. <u>  </u> ) of its array.	0040-033	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The size of leading dimension (ARG NO. <u>  </u> ) of the local array (ARG NO. <u>  </u> ) must be greater than or equal to ( <u>  </u> ).
0040-022	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The order of a matrix (ARG NO. <u>  </u> ) must be less than or equal to the block size (ARG NO. <u>  </u> ).		
0040-023	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> (ARG NO. <u>  </u> ) must be a multiple of the product of (ARG NO. <u>  </u> ) and the number of processes ( <u>  </u> ).		



0040-034	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> SIDE (ARG NO. <u>  </u> ), which specifies whether the input matrix (ARG NO. <u>  </u> ) appears on the left or right of the other input matrix, must be 'L' or 'R'.	0040-044	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The size of the leading dimension, LLD <sub><u>  </u></sub> (element 9 of ARG NO. <u>  </u> ) of the local array (ARG NO. <u>  </u> ) must be greater than zero.
0040-035	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The number of right hand sides (ARG NO. <u>  </u> ) must be greater than or equal to zero.	0040-045	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The size of leading dimension, LLD <sub><u>  </u></sub> (element 9 of ARG NO. <u>  </u> ) of the local array (ARG NO. <u>  </u> ) must be greater than or equal to ( <u>  </u> ).
0040-036	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> TRANS (ARG NO. <u>  </u> ), specifies whether an input matrix (ARG NO. <u>  </u> ), its transpose, or its conjugate transpose should be used. TRANS must be 'N', 'T', or 'C'.	0040-046	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The number of rows, M <sub><u>  </u></sub> (element 3 of ARG NO. <u>  </u> ) in the global matrix (ARG NO. <u>  </u> ) must be greater than zero.
0040-037	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> Task has issued a receive for its own broadcast.	0040-047	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The number of columns, N <sub><u>  </u></sub> (element 4 of ARG NO. <u>  </u> ) in the global matrix (ARG NO. <u>  </u> ) must be greater than zero.
0040-038	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> Minimum message id in message id range (element 1 of ARG NO. 3) must be less than the maximum message id (element 2 of ARG NO. 3).	0040-048	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The global row index (ARG NO. <u>  </u> ) of matrix (ARG NO. <u>  </u> ) must be greater than 0.
0040-039	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The communications context (ARG NO. <u>  </u> ) is invalid.	0040-049	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The global column index (ARG NO. <u>  </u> ) of matrix (ARG NO. <u>  </u> ) must be greater than 0.
0040-040	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The process row or column (ARG NO. <u>  </u> ) must be greater than 0.	0040-050	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The stride (ARG NO. <u>  </u> ) for vector (ARG NO. <u>  </u> ) is 1, but the row block size, MB <sub><u>  </u></sub> (element 5 of ARG NO. <u>  </u> ) is not equal to the block size (element <u>  </u> of ARG NO. <u>  </u> ) of the matrix (ARG NO. <u>  </u> ).
0040-041	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The process row, RSRC <sub><u>  </u></sub> (element 7 of ARG NO. <u>  </u> ) must be greater than or equal to 0 and less than the total number of rows in the process grid.	0040-051	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The row and column block sizes, MB <sub><u>  </u></sub> and NB <sub><u>  </u></sub> (elements 5 and 6 of ARG NO. <u>  </u> ) of the matrix (ARG NO. <u>  </u> ) must be equal.
0040-042	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The process column, CSRC <sub><u>  </u></sub> (element 8 of ARG NO. <u>  </u> ) must be greater than or equal to 0 and less than the total number of columns in the process grid.	0040-052	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The submatrix referenced is incompatible with the global matrix definition. The global row index (ARG NO. <u>  </u> ) plus the number of rows (ARG NO. <u>  </u> ) of the matrix (ARG NO. <u>  </u> ) minus 1 must be less than or equal to the number of rows, M <sub><u>  </u></sub> (element 3 of ARG NO. <u>  </u> ).
0040-043	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The communications context, CTXT <sub><u>  </u></sub> (element 2 of ARG NO. <u>  </u> ) of the matrix (ARG NO. <u>  </u> ) must be equal to the communications context (element 2 of ARG NO. <u>  </u> ) of the matrix (ARG NO. <u>  </u> ).		



0040-053	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The submatrix referenced is incompatible with the global matrix definition. The global column index (ARG NO. <u>  </u> ) plus the number of columns (ARG NO. <u>  </u> ) of the matrix (ARG NO. <u>  </u> ) minus 1 must be less than or equal to the number of columns, N <sub><u>  </u></sub> (element 4 of ARG NO. <u>  </u> ).	0040-061	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The vector (ARG NO. <u>  </u> ) is column-distributed but the row block size, MB <sub><u>  </u></sub> (element 5 of ARG NO. <u>  </u> ) is not equal to the column block size, NB <sub><u>  </u></sub> (element 6 of ARG NO. <u>  </u> ) of the matrix (ARG NO. <u>  </u> ).
0040-055	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The vector (ARG NO. <u>  </u> ) is row-distributed but the column block size, NB <sub><u>  </u></sub> (element 6 of ARG NO. <u>  </u> ) is not equal to the row block size, MB <sub><u>  </u></sub> (element 5 of ARG NO. <u>  </u> ) of the matrix (ARG NO. <u>  </u> ).	0040-062	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The vector (ARG NO. <u>  </u> ) is column-distributed, but the block row offset of the vector is not equal to the block column offset of the matrix (ARG NO. <u>  </u> ).
0040-056	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The vector (ARG NO. <u>  </u> ) is column-distributed but the row block size, MB <sub><u>  </u></sub> (element 5 of ARG NO. <u>  </u> ) is not equal to the row block size, MB <sub><u>  </u></sub> (element 5 of ARG NO. <u>  </u> ) of the matrix (ARG NO. <u>  </u> ).	0040-063	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The vector (ARG NO. <u>  </u> ) is column-distributed, but the block row offset of the vector is not equal to the block row offset of the matrix (ARG NO. <u>  </u> ).
0040-057	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The vector (ARG NO. <u>  </u> ) is row-distributed, but the block column offset of the vector is not equal to the block row offset of the matrix (ARG NO. <u>  </u> ).	0040-064	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The vector (ARG NO. <u>  </u> ) is column-distributed, but the process row ( <u>  </u> ), containing the first element of the vector is not equal to the process row ( <u>  </u> ) containing the first row of the submatrix (ARG NO. <u>  </u> ).
0040-058	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The vector (ARG NO. <u>  </u> ) is row-distributed but the column block size, NB <sub><u>  </u></sub> (element 6 of ARG NO. <u>  </u> ) is not equal to the column block size, NB <sub><u>  </u></sub> (element 6 of ARG NO. <u>  </u> ) of the matrix (ARG NO. <u>  </u> ).	0040-065	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The stride (ARG NO. <u>  </u> ) for vector (ARG NO. <u>  </u> ) must be equal to either 1 or the number of rows, M <sub><u>  </u></sub> (element 3 of ARG NO. <u>  </u> ).
0040-059	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The vector (ARG NO. <u>  </u> ) is row-distributed, but the block column offset of the vector is not equal to the block column offset of the matrix (ARG NO. <u>  </u> ).	0040-066	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The calculated block row offset and block column offset of the submatrix referenced within the global matrix (ARG NO. <u>  </u> ) must be equal.
0040-060	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The vector (ARG NO. <u>  </u> ) is row-distributed, but the process column ( <u>  </u> ), containing the first element of the vector is not equal to the process column ( <u>  </u> ) containing the first column of the submatrix (ARG NO. <u>  </u> ).	0040-067	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> Matrices (ARG NO. <u>  </u> ) and (ARG NO. <u>  </u> ) have incompatible block sizes. The block size (element <u>  </u> of ARG NO. <u>  </u> ) must be equal to the block size (element <u>  </u> of ARG NO. <u>  </u> ).
		0040-068	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The global row index (ARG NO. <u>  </u> ) and global column index (ARG NO. <u>  </u> ) of matrix (ARG NO. <u>  </u> ) must be equal.

0040-069	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> (ARG NO. <u>  </u> ), which represents a process row or column, must be greater than or equal to zero and less than (ARG NO. <u>  </u> ).	0040-080	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The submatrix referenced must be a block column matrix. The block column offset plus the number of columns (ARG NO. <u>  </u> ) of the matrix (ARG NO. <u>  </u> ) must be less than or equal to the column block size (element 6 of ARG NO. <u>  </u> ).
0040-070	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The transform length (ARG NO. <u>  </u> ) must be divisible by the number of tasks ( <u>  </u> ).	0040-081	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> In the process grid, the process row ( <u>  </u> ), containing the first row of the submatrix (ARG NO. <u>  </u> ) must be equal to the process row ( <u>  </u> ) containing the first row of the submatrix (ARG NO. <u>  </u> ).
0040-071	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The transform length (ARG NO. <u>  </u> ) divided by the number of tasks must be an even number.	0040-082	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The communications context, CTXT_ <u>  </u> (element 2 of ARG NO. <u>  </u> ) of the matrix (ARG NO. <u>  </u> ) must be equal to the communications context (element 2 of ARG NO. <u>  </u> ) of the matrix (ARG NO. <u>  </u> ).
0040-072	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The scaling parameter (ARG NO. <u>  </u> ) must be nonzero.	0040-083	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> In the process grid, the process column ( <u>  </u> ), containing the first column of the submatrix (ARG NO. <u>  </u> ) must be equal to the process column ( <u>  </u> ) containing the first column of the submatrix (ARG NO. <u>  </u> ).
0040-073	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The transform length (ARG NO. <u>  </u> ) is not an allowed value. The next higher value is ( <u>  </u> ).	0040-084	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The dimension (ARG NO. <u>  </u> ) of the matrices must be greater than or equal to zero.
0040-074	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The output data distribution format (element 2 of ARG NO. <u>  </u> ) must be zero or one.	0040-085	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The submatrices referenced must be properly aligned. The block offset for matrix (ARG NO. <u>  </u> ) generated by (ARG NO. <u>  </u> ) and block size (element <u>  </u> of ARG NO. <u>  </u> ) must be equal to the block offset for matrix (ARG NO. <u>  </u> ) generated by (ARG NO. <u>  </u> ) and block size (element <u>  </u> of ARG NO. <u>  </u> ).
0040-075	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> (Element <u>  </u> of ARG NO. <u>  </u> ) must be either zero or greater than or equal to the transform dimension (ARG NO. <u>  </u> ).	0040-086	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The communications context ( <u>  </u> ) is not currently active.
0040-076	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The transform direction parameter (ARG NO. <u>  </u> ) must be nonzero.	0040-087	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The communications context ( <u>  </u> ) is invalid.
0040-077	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The transform length (ARG NO. <u>  </u> ) must be less than or equal to ( <u>  </u> ) .		
0040-078	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> (ARG NO. <u>  </u> ) must be nonzero.		
0040-079	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The submatrix referenced must be a block row matrix. The block row offset plus the number of rows (ARG NO. <u>  </u> ) of the matrix (ARG NO. <u>  </u> ) must be less than or equal to the row block size (element 5 of ARG NO. <u>  </u> ).		

0040-088	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The process grid must be defined with the number of rows set to 1.
0040-089	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The vectors referenced must be distributed along the same axis. Either the stride (ARG NO. <u>  </u> ) for vector (ARG NO. <u>  </u> ) and the stride (ARG NO. <u>  </u> ) for vector (ARG NO. <u>  </u> ) must both be equal to 1 or the stride for vector (ARG NO. <u>  </u> ) must be equal to the number of rows, M <sub>  </sub> (element 3 of ARG NO. <u>  </u> ) and the stride for vector (ARG NO. <u>  </u> ) must be equal to the number of rows, M <sub>  </sub> (element 3 of ARG NO. <u>  </u> ).
0040-090	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The row block size, MB <sub>  </sub> (element 5 of ARG NO. <u>  </u> ) of the matrix (ARG NO. <u>  </u> ) must be greater than zero.
0040-091	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The column block size, NB <sub>  </sub> (element 6 of ARG NO. <u>  </u> ) of the matrix (ARG NO. <u>  </u> ) must be greater than zero.
0040-092	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The submatrix referenced must be aligned on a row block boundary. (ARG NO. <u>  </u> ) minus 1 must be a multiple of the row block size, MB <sub>  </sub> (element 5 of ARG NO. <u>  </u> ) of matrix (ARG NO. <u>  </u> ).
0040-093	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The submatrix referenced must be aligned on a column block boundary. (ARG NO. <u>  </u> ) minus 1 must be a multiple of the column block size, NB <sub>  </sub> (element 6 of ARG NO. <u>  </u> ) of matrix (ARG NO. <u>  </u> ).
0040-094	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The global row index (ARG NO. <u>  </u> ) of vector (ARG NO. <u>  </u> ) must be greater than 0 and less than the number of rows, M <sub>  </sub> (element 3 of ARG NO. <u>  </u> ).
0040-095	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The global column index (ARG NO. <u>  </u> ) of vector (ARG NO. <u>  </u> ) must be greater than 0 and less than or equal to the number of columns, N <sub>  </sub> (element 4 of ARG NO. <u>  </u> ).

0040-096	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> TRANS (ARG NO. <u>  </u> ), which specifies the operation to be performed, must be 'T' or 'C'.
0040-097	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The global row index (ARG NO. <u>  </u> ) of matrix (ARG NO. <u>  </u> ) must be greater than 0 and less than or equal to the number of rows in the global matrix, M <sub>  </sub> (element 3 of ARG NO. <u>  </u> ).
0040-098	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The global column index (ARG NO. <u>  </u> ) of matrix (ARG NO. <u>  </u> ) must be greater than 0 and less than or equal to the number of columns in the global matrix, N <sub>  </sub> (element 4 of ARG NO. <u>  </u> ).
0040-099	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The number of rows, M <sub>  </sub> (element 3 of ARG NO. <u>  </u> ) in a null matrix (ARG NO. <u>  </u> ) must be greater than or equal to zero.
0040-100	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The number of columns, N <sub>  </sub> (element 4 of ARG NO. <u>  </u> ) in a null matrix (ARG NO. <u>  </u> ) must be greater than or equal to zero.
0040-101	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The number of rows (ARG NO. <u>  </u> ) of a matrix must be the same for all processes.
0040-102	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The number of columns (ARG NO. <u>  </u> ) of a matrix must be the same for all processes.
0040-103	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The order (ARG NO. <u>  </u> ) of a matrix must be the same for all processes.
0040-104	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The global row index (ARG NO. <u>  </u> ) of the matrix (ARG NO. <u>  </u> ) must be the same for all processes.
0040-105	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The global column index (ARG NO. <u>  </u> ) of the matrix (ARG NO. <u>  </u> ) must be the same for all processes.

0040-106	Context(_) Task(_) Process(,_) Grid _ x _ UPLO (ARG NO. _), which specifies whether an input matrix (ARG NO. _) is upper or lower, must be the same for all processes.	0040-115	Context(_) Task(_) Process(,_) Grid _ x _ The process row RSRC_ (element 7 of ARG NO. _) must be the same for all processes.
0040-107	Context(_) Task(_) Process(,_) Grid _ x _ TRANS (ARG NO. _), which specifies whether an input matrix, its transpose, or its conjugate transpose should be used, must be the same for all processes.	0040-116	Context(_) Task(_) Process(,_) Grid _ x _ The process column CSRC_ (element 8 of ARG NO. _) must be the same for all processes.
0040-108	Context(_) Task(_) Process(,_) Grid _ x _ NRHS (ARG NO. _), which specifies the number of right hand sides in the system to be solved, must be the same for all processes.	0040-117	Context(_) Task(_) Process(,_) Grid _ x _ The number of elements (ARG NO. _) in a work array (ARG NO. _) must be zero, to indicate dynamic allocation, minus one, to indicate workspace query, or greater than or equal to ( ) if a work array is being supplied.
0040-109	Context(_) Task(_) Process(,_) Grid _ x _ ILO (ARG NO. _), which specifies a lower range of rows or columns in a matrix, must be the same for all processes.	0040-118	Context(_) Task(_) Process(,_) Grid _ x _ ILO (ARG NO. _), which specifies a lower range of rows or columns in a matrix, must be greater than or equal to one and less than or equal to the larger of one and the order (ARG NO. _) of the matrix.
0040-110	Context(_) Task(_) Process(,_) Grid _ x _ IHI (ARG NO. _), which specifies an upper range of rows or columns in a matrix, must be the same for all processes.	0040-119	Context(_) Task(_) Process(,_) Grid _ x _ IHI (ARG NO. _), which specifies an upper range of rows or columns in a matrix, must be greater than or equal to the smaller of ILO (ARG NO. _) and the order (ARG NO. _) of the matrix and less than or equal to the order (ARG NO. _) of the matrix.
0040-111	Context(_) Task(_) Process(,_) Grid _ x _ The number of rows, M_ (element 3 of ARG NO. _) in the global matrix (ARG NO. _) must be the same for all processes.	0040-120	Context(_) Task(_) Process(,_) Grid _ x _ The row-distributed vector referenced is incompatible with the global matrix definition. The global column index (ARG NO. _) plus the number of columns (ARG NO. _) of the vector (ARG NO. _) minus 1 must be less than or equal to the number of columns, N_ (element 4 of ARG NO. _).
0040-112	Context(_) Task(_) Process(,_) Grid _ x _ The number of columns, N_ (element 4 of ARG NO. _) in the global matrix (ARG NO. _) must be the same for all processes.	0040-121	Context(_) Task(_) Process(,_) Grid _ x _ The column-distributed vector referenced is incompatible with the global matrix definition. The global row index (ARG NO. _) plus the number of rows (ARG NO. _) of the vector (ARG NO. _) minus 1 must be less than or equal to the number of rows, M_ (element 3 of ARG NO. _).
0040-113	Context(_) Task(_) Process(,_) Grid _ x _ The row block size MB_ (element 5 of ARG NO. _) of the global matrix (ARG NO. _) must be the same for all processes.		
0040-114	Context(_) Task(_) Process(,_) Grid _ x _ The column block size NB_ (element 6 of ARG NO. _) of the global matrix (ARG NO. _) must be the same for all processes.		

0040-122	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> JOBZ (ARG NO. <u>  </u> ), which specifies whether or not to compute eigenvectors, must be 'N' or 'V'.
0040-123	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> RANGE (ARG NO. <u>  </u> ), which specifies which eigenvalues to find, must be 'A', 'V', or 'I'.
0040-124	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> VU (ARG NO. <u>  </u> ), which specifies the upper bound of the interval to be searched for eigenvalues, must be greater than VL (ARG NO. <u>  </u> ), which specifies the lower bound of the interval to be searched for eigenvalues.
0040-125	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> IL (ARG NO. <u>  </u> ), which specifies the index of the smallest eigenvalue to be returned, must be greater than or equal to 1.
0040-126	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> IU (ARG NO. <u>  </u> ), which specifies the index of the largest eigenvalue to be returned, must be greater than or equal to the smaller of the order (ARG NO. <u>  </u> ) of the matrix (ARG NO. <u>  </u> ) and IL (ARG NO. <u>  </u> ) and less than or equal to the order of the matrix.
0040-127	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The global row index (ARG NO. <u>  </u> ) of the matrix (ARG NO. <u>  </u> ) must be 1.
0040-128	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The global column index (ARG NO. <u>  </u> ) of the matrix (ARG NO. <u>  </u> ) must be 1.
0040-129	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The global row index (ARG NO. <u>  </u> ) of the matrix (ARG NO. <u>  </u> ) must be equal to the global row index (ARG NO. <u>  </u> ) of the matrix (ARG NO. <u>  </u> ).
0040-130	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The global column index (ARG NO. <u>  </u> ) of the matrix (ARG NO. <u>  </u> ) must be equal to the global column index (ARG NO. <u>  </u> ) of the matrix (ARG NO. <u>  </u> ).

0040-131	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The number of rows, M_ (element 3 of ARG NO. <u>  </u> ) in the global matrix (ARG NO. <u>  </u> ) must be equal to the number of rows, M_ (element 3 of ARG NO. <u>  </u> ) in the global matrix (ARG NO. <u>  </u> ).
0040-132	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The number of columns, N_ (element 4 of ARG NO. <u>  </u> ) in the global matrix (ARG NO. <u>  </u> ) must be equal to the number of columns, N_ (element 4 of ARG NO. <u>  </u> ) in the global matrix (ARG NO. <u>  </u> ).
0040-133	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The process row, RSRC_ (element 7 of ARG NO. <u>  </u> ) must be zero.
0040-134	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The process column, CSRC_ (element 8 of ARG NO. <u>  </u> ) must be zero.
0040-135	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The process row, RSRC_ (element 7 of ARG NO. <u>  </u> ) must be equal to the process row, RSRC_ (element 7 of ARG NO. <u>  </u> ).
0040-136	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> The process column, CSRC_ (element 8 of ARG NO. <u>  </u> ) must be equal to the process column, CSRC_ (element 8 of ARG NO. <u>  </u> ).
0040-137	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> ORFAC (ARG NO. <u>  </u> ), which specifies which eigenvectors should be orthogonalized, must be the same for all processes.
0040-138	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> JOBZ (ARG NO. <u>  </u> ), which specifies whether or not to compute eigenvectors, must be the same for all processes.
0040-139	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> ) Grid <u>  </u> x <u>  </u> RANGE (ARG NO. <u>  </u> ), which specifies which eigenvalues to find, must be the same for all processes.



0040-140	Context(_) Task(_) Process(_,_) Grid _ x _ VL (ARG NO. _), which specifies the lower bound of the interval to be searched for eigenvalues, must be the same for all processes.	0040-148	Context(_) Task(_) Process(_,_) Grid _ x _ MPI is not initialized. The probable cause is that the BLACS have not been initialized.
0040-141	Context(_) Task(_) Process(_,_) Grid _ x _ VU (ARG NO. _), which specifies the upper bound of the interval to be searched for eigenvalues, must be the same for all processes.	0040-149	Context(_) Task(_) Process(_,_) Grid _ x _ The Cartesian grid is not defined. The probable cause is that the BLACS have not been initialized.
0040-142	Context(_) Task(_) Process(_,_) Grid _ x _ IL (ARG NO. _), which specifies the index of the smallest eigenvalue to be returned, must be the same for all processes.	0040-150	Context(_) Task(_) Process(_,_) Grid _ x _ The Cartesian grid is not defined as two-dimensional. The probable cause is that the BLACS have not been initialized.
0040-143	Context(_) Task(_) Process(_,_) Grid _ x _ IU (ARG NO. _), which specifies the index of the largest eigenvalue to be returned, must be the same for all processes.	0040-230	Context(_) Task(_) Process(_,_) Grid _ x _ The process grid must be defined with either the number of process rows or the number of process columns set to 1.
0040-144	Context(_) Task(_) Process(_,_) Grid _ x _ The vector (ARG NO. _) is row-distributed and TRANS (ARG NO. _) is 'T' or 'C', but the process column (,) containing the first element of the vector is not equal to the process column (,) containing the first column of the submatrix (ARG NO. _).	0040-231	Context(_) Task(_) Process(_,_) Grid _ x _ The number of columns, N_, (element 3 of ARG NO. _) in the global matrix (ARG NO. _) must be the same for all processes.
0040-145	Context(_) Task(_) Process(_,_) Grid _ x _ The vector (ARG NO. _) is column-distributed and TRANS (ARG NO. _) is 'N', but the process row (,) containing the first element of the vector is not equal to the process row (,) containing the first row of the submatrix (ARG NO. _).	0040-232	Context(_) Task(_) Process(_,_) Grid _ x _ The block size (element 4 of ARG NO. _) must be equal to (,).
0040-146	Context(_) Task(_) Process(_,_) Grid _ x _ ABSTOL (ARG NO. _), which specifies the absolute error tolerance for the eigenvalues, must be the same for all processes.	0040-233	Context(_) Task(_) Process(_,_) Grid _ x _ The process row, RSRC_, (element 5 of ARG NO. _) must be equal to (,).
0040-147	Context(_) Task(_) Process(_,_) Grid _ x _ No attributes or key is defined for the communicator. The probable cause is that the BLACS have not been initialized.	0040-234	Context(_) Task(_) Process(_,_) Grid _ x _ The size of leading dimension, LLD_, (element 6 of ARG NO. _) of the local array (ARG NO. _) must be greater than or equal to (,).
		0040-235	Context(_) Task(_) Process(_,_) Grid _ x _ Argument (,) must be greater than or equal to (,).
		0040-236	Context(_) Task(_) Process(_,_) Grid _ x _ DTYPE_ (element 1 of ARG NO. _), which specifies the descriptor type, must be the same for all processes.
		0040-237	Context(_) Task(_) Process(_,_) Grid _ x _ The communications context, CTXT_, (element 2 of ARG NO. _), must be the same for all processes.

0040-238	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The number of elements in the matrix (ARG NO. <u>  </u> ) supplied to store the factor must be greater than or equal to ( <u>  </u> ).
0040-239	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> TRANS (ARG NO. <u>  </u> ), which specifies the operation to be performed, must be 'N' or 'n'.
0040-242	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The number of rows, M <sub><u>  </u></sub> , (element 3 of ARG NO. <u>  </u> ) in the global matrix must be greater than the half-bandwidth, K.
0040-243	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The half bandwidth, K, (ARG NO. <u>  </u> ) must be the same for all processes.
0040-245	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The process row or column, (element 5 of ARG NO. <u>  </u> ) must be equal to ( <u>  </u> ).
0040-248	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The half bandwidth (ARG NO. <u>  </u> ) of a matrix must be greater than or equal to zero.
0040-249	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The half bandwidth (ARG NO. <u>  </u> ) of the band matrix (ARG NO. <u>  </u> ) must be less than the order of the matrix.
0040-250	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The number of rows (ARG NO. <u>  </u> ) of matrix (ARG NO. <u>  </u> ) must be smaller than or equal to the product of the number of processors and the block size (element <u>  </u> of ARG NO. <u>  </u> ) minus the modulus of (ARG NO. <u>  </u> ) minus one with the block size (element <u>  </u> of ARG NO. <u>  </u> ).
0040-272	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The value of UPLO is U. NB <sub><u>  </u></sub> (element <u>  </u> of ARG NO. <u>  </u> ) must be greater than or equal to the half bandwidth, K (ARG NO. <u>  </u> ).
0040-276	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The number of columns, N <sub><u>  </u></sub> , (element 4 of ARG NO. <u>  </u> ) in the global matrix must be greater than or equal to the number of right hand sides (ARG NO. <u>  </u> ).

0040-277	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The global row index (ARG NO. <u>  </u> ) of matrix (ARG NO. <u>  </u> ) must be greater than 0 and less than or equal to the number of rows in the global matrix, M <sub><u>  </u></sub> , (element <u>  </u> of ARG NO. <u>  </u> ).
0040-278	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The global column index (ARG NO. <u>  </u> ) of matrix (ARG NO. <u>  </u> ) must be greater than 0 and less than or equal to the number of columns in the global matrix, N <sub><u>  </u></sub> , (element <u>  </u> of ARG NO. <u>  </u> ).
0040-279	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The block size (element 4 of ARG NO. <u>  </u> ) must be the same for all processes.
0040-280	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The process column CSRC <sub><u>  </u></sub> , (element 5 of ARG NO. <u>  </u> ) must be the same for all processes.
0040-281	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The number of columns, N <sub><u>  </u></sub> , (element 3 of ARG NO. <u>  </u> ) in the global matrix (ARG NO. <u>  </u> ) must be greater than zero.
0040-282	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The process row RSRC <sub><u>  </u></sub> , (element 5 of ARG NO. <u>  </u> ) must be the same for all processes.
0040-286	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The descriptor type, DTYPE <sub><u>  </u></sub> (element 1 of ARG NO. <u>  </u> ) is invalid. The valid descriptor type for this routine is <u>  </u> .
0040-287	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The submatrix referenced is incompatible with the global matrix definition. The global column index (ARG NO. <u>  </u> ) plus the number of columns (ARG NO. <u>  </u> ) of the matrix (ARG NO. <u>  </u> ) minus 1 must be less than or equal to the number of columns, N <sub><u>  </u></sub> , (element 3 of ARG NO. <u>  </u> ).
0040-289	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The row block size, MB <sub><u>  </u></sub> , (element 4 of ARG NO. <u>  </u> ) of the matrix (ARG NO. <u>  </u> ) must be greater than zero.

0040-290	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The number of columns, N <sub><u>  </u></sub> (element 3 of ARG NO. <u>  </u> ) in a null matrix (ARG NO. <u>  </u> ) must be greater than or equal to zero.
0040-291	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The column block size, NB <sub><u>  </u></sub> (element 4 of ARG NO. <u>  </u> ) of the matrix (ARG NO. <u>  </u> ) must be greater than zero.
0040-292	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The order (ARG NO. <u>  </u> ) of matrix (ARG NO. <u>  </u> ) must be smaller than or equal to the product of the number of processors and the block size (element <u>  </u> of ARG NO. <u>  </u> ) minus the modulus of (ARG NO. <u>  </u> ) minus one with the block size (element <u>  </u> of ARG NO. <u>  </u> ).
0040-293	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The descriptor type, DTYPE <sub><u>  </u></sub> (element 1 of ARG NO. <u>  </u> ) is invalid. Valid descriptor types for this routine are <u>  </u> and <u>  </u> .

0040-294	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The descriptor type, DTYPE <sub><u>  </u></sub> (element 1 of ARG NO. <u>  </u> ) is invalid. Valid descriptor types for this routine are <u>  </u> , <u>  </u> and <u>  </u> .
0040-295	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> End of input-argument error reporting. The first argument found to have an invalid value was (ARG NO. <u>  </u> ). Parallel ESSL has returned a valid array descriptor (ARG NO. <u>  </u> ).
0040-297	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> End of global input-argument error reporting. For more information, refer to Parallel ESSL Guide and Reference.
0040-299	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> End of input-argument error reporting. For more information, refer to Parallel ESSL Guide and Reference.

**Note:** There are more input-argument error messages listed in “Input-Argument Error Messages (800-999)” on page 122.

## Computational Error Messages (300-399)

0040-300	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The input matrix (ARG NO. <u>  </u> ) is singular. The first diagonal element found to be exactly 0 was in column ( <u>  </u> ).	0040-305	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> No eigenvalues were computed since the Gershgorin interval initially used was incorrect.
0040-301	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The storage space, specified by (ARG NO. <u>  </u> ) is insufficient. ( <u>  </u> ) bytes are required.	0040-306	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> ( <u>  </u> ) eigenvectors failed to converge after ( <u>  </u> ) iterations. The indices are stored in IFAIL (ARG NO. <u>  </u> ).
0040-302	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The matrix (ARG NO. <u>  </u> ) is not positive definite. The leading minor of order ( <u>  </u> ) has a nonpositive determinant.	0040-307	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> Eigenvectors corresponding to one or more clusters of eigenvalues could not be reorthogonalized because of insufficient workspace. The indices of the clusters are stored in ICLUSTER (ARG NO. <u>  </u> ).
0040-303	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> Bisection failed to converge for some eigenvalues. The eigenvalues may not be as accurate as the absolute and relative tolerances.	0040-308	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> All of the eigenvectors between VL (ARG NO. <u>  </u> ) and VU (ARG NO. <u>  </u> ) could not be computed due to insufficient workspace. The number of eigenvectors computed is returned in NZ (ARG NO. <u>  </u> ).
0040-304	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The number of eigenvalues computed (ARG NO. <u>  </u> ) does not match the number of eigenvalues requested.		



0040-309	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The number of eigenvalues computed (ARG NO. <u>  </u> ) does not equal the number of eigenvectors computed (ARG NO. <u>  </u> ).
0040-310	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The input matrix is either (nearly) singular or reducible. The value of INFO is ( <u>  </u> ). The portion of the global submatrix stored on process ( <u>  </u> ) and factored locally is either (nearly) singular or reducible. A pivot element whose magnitude is too small or zero was detected.
0040-311	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The input matrix is not diagonally dominant. The value of INFO is ( <u>  </u> ). The portion of the global submatrix stored on process ( <u>  </u> ) and factored locally is not diagonally dominant. A pivot element whose magnitude is too small or zero was detected.
0040-312	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The input matrix is not positive definite. The value of INFO is ( <u>  </u> ). The portion of the global submatrix stored on process ( <u>  </u> ) and factored locally is not positive definite. A pivot element whose value is less than or equal to a small positive number was detected.
0040-313	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The maximum number of specified iterations ( <u>  </u> ) has been performed without satisfying the convergence criterion as specified by ( <u>  </u> ).
0040-314	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The preconditioner, as specified by argument ( <u>  </u> ) for sparse matrix ( <u>  </u> ) is unstable.
0040-315	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The sparse matrix ( <u>  </u> ) contains duplicated coefficients.
0040-316	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The maximum number of specified iterations ( <u>  </u> ) has been performed without satisfying the convergence criterion as specified by (ARG NO. <u>  </u> ).

0040-317	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The preconditioner, as specified by argument (ARG NO. <u>  </u> ) for sparse matrix (ARGS NO. <u>  </u> - <u>  </u> ) is unstable.
0040-318	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The sparse matrix (ARGS NO. <u>  </u> - <u>  </u> ) contains duplicated coefficients.
0040-319	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The sparse matrix ( <u>  </u> ) contains empty row(s).
0040-320	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The sparse matrix (ARGS NO. <u>  </u> - <u>  </u> ) contains empty row(s).
0040-321	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The input matrix is either (nearly) singular or reducible. The value of INFO is ( <u>  </u> ). The portion of the global submatrix stored on process ( <u>  </u> ) representing interactions with other processes is either (nearly) singular or reducible. A pivot element whose magnitude is too small or zero was detected.
0040-322	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The input matrix is not diagonally dominant. The value of INFO is ( <u>  </u> ). The portion of the global submatrix stored on process ( <u>  </u> ) representing interactions with other processes is not diagonally dominant. A pivot element whose value is less than or equal to a small positive number was detected.
0040-323	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The input matrix is not positive definite. The value of INFO is ( <u>  </u> ). The portion of the global submatrix stored on process ( <u>  </u> ) representing interactions with other processes is not positive definite. A pivot element whose value is less than or equal to a small positive number was detected.
0040-324	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The inverse of matrix (ARG NO. <u>  </u> ) could not be computed. The first diagonal element of the factored matrix found to be exactly zero was in column ( <u>  </u> ).

---

0040-325	Context( <u> </u> ) Task( <u> </u> ) Process( <u> </u> , <u> </u> ) Grid <u> </u> x <u> </u> The singular values of matrix (ARG NO. <u> </u> ) failed to converge. ( <u> </u> ) elements of the superdiagonal of an intermediate bidiagonal form failed to converge to zero.
----------	---

---

0040-326	Context( <u> </u> ) Task( <u> </u> ) Process( <u> </u> , <u> </u> ) Grid <u> </u> x <u> </u> The singular values (ARG NO. 0) are not the same on all processes.
----------	--

## Resource Error Messages (400-499)

---

0040-400	Context( <u> </u> ) Task( <u> </u> ) Process( <u> </u> , <u> </u> ) Grid <u> </u> x <u> </u> An internal buffer allocation has failed due to insufficient memory.
----------	--

---

0040-401	Context( <u> </u> ) Task( <u> </u> ) Process( <u> </u> , <u> </u> ) Grid <u> </u> x <u> </u> Unable to allocate component(s) of derived data type ( <u> </u> ) due to insufficient memory.
----------	---

## Communication Error Messages (500-599)

---

0040-501	Context( <u> </u> ) Task( <u> </u> ) Process( <u> </u> , <u> </u> ) Grid <u> </u> x <u> </u> During process grid synchronization, task( <u> </u> ) has reported an incorrect grid ( <u> </u> ). Actual grid dimension is ( <u> </u> ).
----------	---

---

0040-502	Context( <u> </u> ) Task( <u> </u> ) Process( <u> </u> , <u> </u> ) Grid <u> </u> x <u> </u> During process grid synchronization, task( <u> </u> ) has returned an incorrect value
----------	---

( ) for its own task id.

---

0040-503	Context( <u> </u> ) Task( <u> </u> ) Process( <u> </u> , <u> </u> ) Grid <u> </u> x <u> </u> During process grid synchronization, task( <u> </u> ) has returned an incorrect message id range ( <u> </u> ). Expected range is ( <u> </u> ).
----------	--

---

## Informational and Attention Messages (600-699)

---

0040-600	Context( <u> </u> ) Task( <u> </u> ) Process( <u> </u> , <u> </u> ) Grid <u> </u> x <u> </u> Attention: process is sending data to itself.
----------	---

---

0040-601	Context( <u> </u> ) Task( <u> </u> ) Process( <u> </u> , <u> </u> ) Grid <u> </u> x <u> </u> Attention: process is receiving data from itself.
----------	---

---

0040-602	Context( <u> </u> ) Task( <u> </u> ) Process( <u> </u> , <u> </u> ) Grid <u> </u> x <u> </u> Attention: process has received data whose length ( <u> </u> ) differs from the expected length ( <u> </u> ).
----------	---

---

0040-603	Context( <u> </u> ) Task( <u> </u> ) Process( <u> </u> , <u> </u> ) Grid <u> </u> x <u> </u> Attention: The message id range for point-to-point communications will be reused every ( <u> </u> ) messages.
----------	---

---

0040-604	Context( <u> </u> ) Task( <u> </u> ) Process( <u> </u> , <u> </u> ) Grid <u> </u> x <u> </u> Attention: The message id range for scoped communications will be reused every ( <u> </u> ) operations.
----------	---

---

0040-605	Context( <u> </u> ) Task( <u> </u> ) Process( <u> </u> , <u> </u> ) Grid <u> </u> x <u> </u> Attention: gettimeofday system call returned bad rc ( <u> </u> ).
----------	---

---

0040-606	Context( <u> </u> ) Task( <u> </u> ) Process( <u> </u> , <u> </u> ) Grid <u> </u> x <u> </u> Attention: Attempt to change message id range after process grid definition. Message id range not changed.
----------	--

---

0040-607	Context( <u> </u> ) Task( <u> </u> ) Process( <u> </u> , <u> </u> ) Grid <u> </u> x <u> </u> Attention: Configuration parameter (ARG NO. <u> </u> ) is invalid.
----------	--

---

0040-608	Context( <u> </u> ) Task( <u> </u> ) Process( <u> </u> , <u> </u> ) Grid <u> </u> x <u> </u> Attention: Application waited for memory allocation.
----------	--

---

0040-609	Context( <u> </u> ) Task( <u> </u> ) Process( <u> </u> , <u> </u> ) Grid <u> </u> x <u> </u> Attention: getrusage system call returned bad rc ( <u> </u> ).
----------	--

---

0040-610	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> Attention: Attempt to define process grid before calling BLACS_GET.
0040-611	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> Attention: BLACS system context can only be set by calling BLACS_GET.
0040-612	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> Attention: The number of rings cannot be set to zero.
0040-613	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> Attention: The number of branches ( <u>  </u> ) must be greater than zero.
0040-614	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> Attention: Cannot set BLACS debug level.

0040-615	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> Attention: Environment variable PESSL_DESC_TYPE has specified the use of obsolete descriptor vectors.
0040-616	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> Convergence indicator for iterative method ( <u>  </u> ) at step ( <u>  </u> ): ( <u>  </u> ).
0040-617	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> Message buffer space exceeded for error message number ( <u>  </u> ). One or more instances of the message was suppressed.
0040-618	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> Performance may be degraded due to limited buffer space availability.

## Miscellaneous Error Messages (700-799)

0040-700	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> Internal Parallel ESSL error number ( <u>  </u> ). Contact your IBM service representative.
0040-701	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> Unable to open Parallel ESSL Message Catalog. See your System Administrator for further assistance.
0040-702	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> Internal Parallel ESSL error: message buffer space exceeded for error message number ( <u>  </u> ). Contact your IBM service representative.
0040-703	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> Internal Parallel ESSL error: message number requested ( <u>  </u> ) is outside of the valid range. Contact your IBM service representative.
0040-704	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> Parallel ESSL has been called from outside the process grid definition.
0040-705	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The communications context ( <u>  </u> ) is invalid.
0040-706	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The process grid must be defined with the number of rows set to 1.

0040-707	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The process grid must be defined with the number of columns set to 1.
0040-708	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The user supplied subroutine ( <u>  </u> ) has produced an incorrect output. Argument ( <u>  </u> ) must be greater than or equal to 1 and less than or equal to the number of processes in the process grid.
0040-709	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The user supplied subroutine ( <u>  </u> ) has produced an incorrect output. Argument ( <u>  </u> ) must be greater than or equal to 0 and less than the the number of processes in the process grid.
0040-710	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The user supplied subroutine (ARG NO. <u>  </u> ) has produced an incorrect output. (ARG NO. <u>  </u> ) of the user supplied subroutine must be greater than or equal to 1 and less than or equal to the number of processes in the process grid.
0040-711	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The user supplied subroutine (ARG NO. <u>  </u> ) has produced an incorrect output. (ARG NO. <u>  </u> ) of the user supplied subroutine must be greater than or equal to 0 and less than the number of processes in the process grid.

---

0040-712 Context() Task() Process(,) Grid \_ x \_  
The size of input array (ARG NO. \_) must be greater than or equal to (\_).

---

0040-713 Context() Task() Process(,) Grid \_ x \_  
Environment variable  
PESSL\_DESC\_TYPE has specified the use of obsolete descriptor vectors. You must update the descriptor vectors in

---

your program as described in the Parallel ESSL Guide and Reference.

---

0040-799 Context() Task() Process(,) Grid \_ x \_  
Unable to locate message number (.). Please refer to the chapter entitled 'Using Error Handling' in the Parallel ESSL Guide and Reference (SA22-7906) for the full message text.

---

## Input-Argument Error Messages (800-999)

---

0040-800 Context() Task() Process(,) Grid \_ x \_  
DTYPE\_ (element \_ of ARG NO. \_) for matrix (ARG NO. \_) is \_. The process grid for matrix (ARG NO. \_) must be defined with the number of rows set to 1.

---

0040-801 Context() Task() Process(,) Grid \_ x \_  
DTYPE\_ (element \_ of ARG NO. \_) for matrix (ARG NO. \_) is \_. The process grid for matrix (ARG NO. \_) must be defined with the number of columns set to 1.

---

0040-802 Context() Task() Process(,) Grid \_ x \_  
The global column index, (ARG NO. \_) must be equal to the global row index (ARG NO. \_).

---

0040-803 Context() Task() Process(,) Grid \_ x \_  
The submatrix referenced is incompatible with the global matrix definition. The global row index (ARG NO. \_) plus the number of rows (ARG NO. \_) of the matrix (ARG NO. \_) minus 1 must be less than or equal to the number of rows, N\_ (element \_ of ARG NO. \_).

---

0040-804 Context() Task() Process(,) Grid \_ x \_  
The number of rows, M\_ (element \_ of ARG NO. \_) in the global matrix (ARG NO. \_) must be equal to one.

---

0040-805 Context() Task() Process(,) Grid \_ x \_  
The number of columns, N\_ (element \_ of ARG NO. \_) in the global matrix (ARG NO. \_) must be equal to one.

---

0040-806 Context() Task() Process(,) Grid \_ x \_  
DTYPE\_ (element \_ of ARG NO. \_) is one. At least one of the following must be true: For the global matrix (ARG NO. \_), the number of rows, M\_ (element \_ of ARG NO. \_) must be equal to one or

---

the number of columns, N\_ (element \_ of ARG NO. \_) must be equal to one.

---

0040-807 Context() Task() Process(,) Grid \_ x \_  
The row block size MB\_ (element \_ of ARG NO. \_) of the global matrix (ARG NO. \_) must be the same for all processes.

---

0040-816 Context() Task() Process(,) Grid \_ x \_  
The process grid must be defined with the number of columns set to 1.

---

0040-817 Context() Task() Process(,) Grid \_ x \_  
The size of array (ARG NO. \_) must be greater than or equal to (\_).

---

0040-818 Context() Task() Process(,) Grid \_ x \_  
The array descriptor ( ) has not been initialized. Routine ( ) must be called prior to this routine.

---

0040-819 Context() Task() Process(,) Grid \_ x \_  
The array descriptor ( ) contains invalid component(s). Routine ( ) must be called prior to this routine.

---

0040-820 Context() Task() Process(,) Grid \_ x \_  
The array descriptor ( ) contains invalid component(s).

---

0040-821 Context() Task() Process(,) Grid \_ x \_  
The pointer(s) specified by argument ( ) are not associated and therefore cannot be freed.

---

0040-822 Context() Task() Process(,) Grid \_ x \_  
The size of array ( ) must be greater than or equal to (\_). Routine ( ) must be called prior to this routine.

---

0040-823 Context() Task() Process(,) Grid \_ x \_  
The sparse matrix ( ) is invalid. Routine ( ) must be called prior to this routine.

---

0040-824	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The sparse matrix ( <u>  </u> ) was not initialized properly. Some local row(s) are missing. Additional calls to ( <u>  </u> ) may be required.	0040-836	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> Argument ( <u>  </u> ) must be greater than or equal to ( <u>  </u> ) and less than or equal to ( <u>  </u> ).
0040-825	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The value of argument ( <u>  </u> ) is ( <u>  </u> ); therefore argument ( <u>  </u> ) is required.	0040-837	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The sparse matrix (ARGS NO. <u>  </u> - <u>  </u> ) was not initialized properly. Some local row(s) are missing. Additional calls to ( <u>  </u> ) may be required.
0040-826	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The storage format ( <u>  </u> ) specified for sparse matrix ( <u>  </u> ) must be ( <u>  </u> ).	0040-838	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The storage format (ARG NO. <u>  </u> ) specified for sparse matrix (ARGS NO. <u>  </u> - <u>  </u> ) must be ( <u>  </u> ).
0040-827	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> Argument ( <u>  </u> ) must be equal to ( <u>  </u> ).	0040-839	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> (ARG NO. <u>  </u> ) must be equal to ( <u>  </u> ).
0040-828	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The storage format for sparse matrix ( <u>  </u> ) must be ( <u>  </u> ). Routine ( <u>  </u> ) must be called prior to this routine.	0040-840	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> (ARG NO. <u>  </u> ) must be the same for all processes.
0040-829	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The contents of array descriptor (ARG NO. <u>  </u> ) are invalid.	0040-841	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> Element ( <u>  </u> ) of array (ARG NO. <u>  </u> ) must be equal to ( <u>  </u> ).
0040-830	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The sparse matrix (ARGS NO. <u>  </u> - <u>  </u> ) is invalid. Routine ( <u>  </u> ) must be called prior to this routine.	0040-842	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> Element ( <u>  </u> ) of array (ARG NO. <u>  </u> ) must be greater than or equal to ( <u>  </u> ) and less than or equal to ( <u>  </u> ).
0040-831	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> (ARG NO. <u>  </u> ) must be greater than or equal to ( <u>  </u> ) and less than or equal to ( <u>  </u> ).	0040-843	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The process row, RSRC_ <u>  </u> , (element <u>  </u> of ARG NO. <u>  </u> ) must be greater than or equal to 0 and less than the total number of rows in the process grid.
0040-832	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The storage format of sparse matrix (ARGS NO. <u>  </u> - <u>  </u> ) is invalid.	0040-844	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The process column, CSRC_ <u>  </u> , (element <u>  </u> of ARG NO. <u>  </u> ) must be greater than or equal to 0 and less than the total number of columns in the process grid.
0040-833	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> One or more of the rows requested for insertion with ( <u>  </u> ) does not belong to this process.	0040-845	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The process column, CSRC_ <u>  </u> , (element <u>  </u> of ARG NO. <u>  </u> ) must be equal to the process row, RSRC_ <u>  </u> , (element <u>  </u> of ARG NO. <u>  </u> ).
0040-834	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> One or more of the rows requested for insertion with (ARG NO. <u>  </u> ) does not belong to this process.	0040-846	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The workspace size (ARG NO. <u>  </u> ) has been specified as minus one for a subset of the processes and therefore it must be specified as minus one for all processes.
0040-835	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> Argument ( <u>  </u> ) must be the same for all processes.		



0040-847	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The preconditioner ( <u>  </u> ) contains invalid components. Routine ( <u>  </u> ) must be called prior to this routine.	0040-858	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The global row index (ARG NO. <u>  </u> ) of matrix (ARG NO. <u>  </u> ) must be greater than 0 and less than or equal to the number of columns in the global matrix, N <sub><u>  </u></sub> (element <u>  </u> of ARG NO. <u>  </u> ).
0040-848	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The preconditioner (ARG NO. <u>  </u> ) contains invalid components. Routine ( <u>  </u> ) must be called prior to this routine.	0040-859	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The process row, RSRC <sub><u>  </u></sub> (element <u>  </u> of ARG NO. <u>  </u> ) must be equal to the process row, RSRC <sub><u>  </u></sub> (element <u>  </u> of ARG NO. <u>  </u> ).
0040-849	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The size of array ( <u>  </u> ) must be greater than or equal to ( <u>  </u> ) and less than or equal to ( <u>  </u> ).	0040-860	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> TRANS (ARG NO. <u>  </u> ), which specifies the computation to be performed, must be 'N' or 'T'.
0040-850	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The matrix type ( <u>  </u> ) specified for sparse matrix ( <u>  </u> ) must be ( <u>  </u> ).	0040-861	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> TRANS (ARG NO. <u>  </u> ), which specifies the computation to be performed, must be 'N' or 'C'.
0040-851	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The matrix type (ARG NO. <u>  </u> ) specified for sparse matrix (ARGS NO. <u>  </u> - <u>  </u> ) must be ( <u>  </u> ).	0040-862	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> NORM (ARG NO. <u>  </u> ), which specifies whether to calculate the 1-norm condition number or the infinity-norm condition number, must be '1', 'O', or 'T'.
0040-852	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> Element ( <u>  </u> ) of vector (ARG NO. <u>  </u> ) must be greater than or equal to zero.	0040-863	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> ANORM (ARG NO. <u>  </u> ), which specifies the norm of the matrix, must be the same for all processes.
0040-853	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> Element ( <u>  </u> ) of vector (ARG NO. <u>  </u> ) must be the same for all processes.	0040-864	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> NORM (ARG NO. <u>  </u> ), which specifies which condition number is required, must be the same for all processes.
0040-854	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> LWORK (ARG NO. <u>  </u> ), which specifies the size of the local work array, must be the same for all processes.	0040-865	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> NORM (ARG NO. <u>  </u> ), which specifies the computation to be performed, must be 'M', '1', 'O', 'T', 'F', or 'E'.
0040-855	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The preconditioner data structure ( <u>  </u> ) must be passed unchanged to the solver subroutine.	0040-866	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> IBTYPE (ARG NO. <u>  </u> ), which specifies the problem type, must be 1, 2, or 3.
0040-856	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The preconditioner data structure (ARG NO. <u>  </u> ) must be passed unchanged to the solver subroutine.	0040-867	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> IBTYPE (ARG NO. <u>  </u> ), which specifies the problem type, must be the same for all processes.
0040-857	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> The size of array ( <u>  </u> ) must be greater than or equal to ( <u>  </u> ).		

0040-868	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> JOBV (ARG NO. <u>  </u> ), which specifies whether or not to compute left singular vectors, must be 'N' or 'V'.
0040-869	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> JOBVT (ARG NO. <u>  </u> ), which specifies whether or not to compute right singular vectors, must be 'N' or 'V'.
0040-870	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> JOBV (ARG NO. <u>  </u> ), which specifies whether or not to compute left singular vectors, must be the same for all processes.
0040-871	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> JOBVT (ARG NO. <u>  </u> ), which specifies whether or not to compute right singular vectors, must be the same for all processes.
0040-872	Context( <u>  </u> ) Task( <u>  </u> ) Process( <u>  </u> , <u>  </u> ) Grid <u>  </u> x <u>  </u> (ARG NO. <u>  </u> ) must be greater than or equal ( <u>  </u> ).





---

## Part 2. Reference Information

This part of the book is organized into seven areas, providing reference information for coding the Parallel ESSL subroutines. It is organized as follows:

- Level 2 PBLAS
- Level 3 PBLAS
- Linear Algebraic Equations
- Eigensystem Analysis and Singular Value Analysis
- Fourier Transforms
- Random Number Generation
- Utilities



---

## Chapter 6. Level 2 PBLAS

This chapter describes the Level 2 PBLAS subroutines.

---

### Overview of the Level 2 PBLAS Subroutines

The Level 2 PBLAS include a subset of the standard set of distributed memory parallel versions of the Level 2 BLAS.

**Note:** These subroutines are designed in accordance with the proposed Level 2 PBLAS standard. (See references [15], [16], and [18].) If these subroutines do not comply with the standard as approved, IBM will consider updating them to do so. If IBM updates these subroutines, the update could require modifications of the calling application program.

*Table 48. List of Level 2 PBLAS*

Descriptive Name	Long-Precision Subprogram	Page
Matrix-Vector Product for a General Matrix or Its Transpose	PDGEMV PZGEMV	131
Matrix-Vector Product for a Real Symmetric or a Complex Hermitian Matrix	PDSYMV PZHEMV	154
Rank-One Update of a General Matrix	PDGER PZGERC PZGERU	168
Rank-One Update of a Real Symmetric or a Complex Hermitian Matrix	PDSYR PZHER	186
Rank-Two Update of a Real Symmetric or a Complex Hermitian Matrix	PDSYR2 PZHER2	197
Matrix-Vector Product for a Triangular Matrix or Its Transpose	PDTRMV PZTRMV	212
Solution of Triangular System of Equations with a Single Right-Hand Sides	PDTRSV PZTRSV	288

---

## Level 2 PBLAS Subroutines

This section contains the Level 2 PBLAS subroutine descriptions.

## PDGEMV and PZGEMV — Matrix-Vector Product for a General Matrix or Its Transpose

### Purpose

PDGEMV computes one of the following matrix-vector products:

- $y \leftarrow \alpha Ax + \beta y$
- $y \leftarrow \alpha A^T x + \beta y$

PZGEMV computes one of the following matrix-vector products:

- $y \leftarrow \alpha Ax + \beta y$
- $y \leftarrow \alpha A^T x + \beta y$
- $y \leftarrow \alpha A^H x + \beta y$

where, in the formulas above:

- $A$  represents the global general submatrix  $A_{ia:ia+m-1, ja:ja+n-1}$ .
- $x$  represents the global vector:
  - For  $transa = 'N'$ :
    - For  $incx = M\_X$ , it is  $X_{ix:ix, jx:jx+n-1}$ .
    - For  $incx = 1$  and  $incx \neq M\_X$ , it is  $X_{ix:ix+n-1, jx:jx}$ .
  - For  $transa = 'T'$  or  $'C'$ :
    - For  $incx = M\_X$ , it is  $X_{ix:ix, jx:jx+m-1}$ .
    - For  $incx = 1$  and  $incx \neq M\_X$ , it is  $X_{ix:ix+m-1, jx:jx}$ .
- $y$  represents the global vector:
  - For  $transa = 'N'$ :
    - For  $incy = M\_Y$ , it is  $Y_{iy:iy, jy:jy+m-1}$ .
    - For  $incy = 1$  and  $incy \neq M\_Y$ , it is  $Y_{iy:iy+m-1, jy:jy}$ .
  - For  $transa = 'T'$  or  $'C'$ :
    - For  $incy = M\_Y$ , it is  $Y_{iy:iy, jy:jy+n-1}$ .
    - For  $incy = 1$  and  $incy \neq M\_Y$ , it is  $Y_{iy:iy+n-1, jy:jy}$ .
- $\alpha$  and  $\beta$  are scalars.

**Note:** No data should be moved to form  $A^T$  or  $A^H$ ; that is, the matrix  $A$  should always be stored in its untransposed form.

In the following three cases, no computation is performed and the subroutine returns after doing some parameter checking:

- $m = 0$
- $n = 0$
- $\alpha$  is zero and  $\beta$  is one.

See references [15] and [16].

Table 49. Data Types

$\alpha, \beta, A, x, y$	Subprogram
Long-precision real	PDGEMV
Long-precision complex	PZGEMV

### Syntax

<b>Fortran</b>	CALL PDGEMV   PZGEMV ( <i>transa</i> , <i>m</i> , <i>n</i> , <i>alpha</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>x</i> , <i>ix</i> , <i>jx</i> , <i>desc_x</i> , <i>incx</i> , <i>beta</i> , <i>y</i> , <i>iy</i> , <i>jy</i> , <i>desc_y</i> , <i>incy</i> )
----------------	--

C and C++	pdgemv   pzgemv ( <i>transa</i> , <i>m</i> , <i>n</i> , <i>alpha</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>x</i> , <i>ix</i> , <i>jx</i> , <i>desc_x</i> , <i>incx</i> , <i>beta</i> , <i>y</i> , <i>iy</i> , <i>jy</i> , <i>desc_y</i> , <i>incy</i> );
-----------	--

## On Entry

*transa* indicates the form of matrix  $A$  to use in the computation, where:

If *transa* = 'N',  $A$  is used in the computation.

If *transa* = 'T',  $A^T$  is used in the computation.

If *transa* = 'C',  $A^H$  is used in the computation.

Scope: **global**

Specified as: a single character; *transa* = 'N', 'T', or 'C'.

*m* is the number of rows in submatrix  $A$  used in the computation, and:

If *transa* = 'N', it is the number of elements in vector  $y$ .

If *transa* = 'T' or 'C', it is the number of elements in vector  $x$ .

Scope: **global**

Specified as: a fullword integer;  $m \geq 0$ .

*n* is the number of columns in submatrix  $A$  used in the computation, and:

If *transa* = 'N', it is the number of elements in vector  $x$ .

If *transa* = 'T' or 'C', it is the number of elements in vector  $y$ .

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

*alpha* is the scalar  $\alpha$ .

Scope: **global**

Specified as: a number of the data type indicated in Table 49 on page 131.

*a* is the local part of the global general matrix  $A$ . This identifies the **first element** of the local array  $A$ . This subroutine computes the location of the first element of the local subarray used, based on *ia*, *ja*, *desc\_a*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading  $\text{LOCp}(ia+m-1)$  by  $\text{LOCq}(ja+n-1)$  part of the local array  $A$  must contain the local pieces of the leading  $ia+m-1$  by  $ja+n-1$  part of the global matrix.

**Note:** No data should be moved to form  $A^T$  or  $A^H$ ; that is, the matrix  $A$  should always be stored in its untransposed form.

Scope: **local**

Specified as: an  $\text{LLD\_A}$  by (at least)  $\text{LOCq}(\text{N\_A})$  array, containing numbers of the data type indicated in Table 49 on page 131. Details about the block-cyclic data distribution of global matrix  $A$  are stored in *desc\_a*.

*ia* is the row index of the global matrix  $A$ , identifying the first row of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq \text{M\_A}$  and  $ia+m-1 \leq \text{M\_A}$ .

*ja* is the column index of the global matrix  $A$ , identifying the first column of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$  and  $ja+n-1 \leq N\_A$ .

*desc\_a* is the array descriptor for global matrix *A*, described in the following table:

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $m = 0$ or $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	If $m = 0$ or $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_A < p$	Global
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < q$	Global
9	LLD_A	The leading dimension of the local array	$LLD\_A \geq \max(1, LOCp(M\_A))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

*x* is the local part of the global matrix *X*. This identifies the **first element** of the local array *X*. This subroutine computes the location of the first element of the local subarray used, based on *ix*, *jx*, *desc\_x*, *p*, *q*, *myrow*, and *mycol*; therefore, assuming the following:

If *transa* = 'N', *numx* = *n*

If *transa* = 'T' or 'C', *numx* = *m*

the following must be true:

- If *incx* = *M\_X*, the leading *LOCp(ix)* by *LOCq(jx+numx-1)* part of the local array *X* must contain the local pieces of the leading *ix* by *jx+numx-1* part of the global matrix.
- If *incx* = 1 and *incx*  $\neq$  *M\_X*, the leading *LOCp(ix+numx-1)* by *LOCq(jx)* part of the local array *X* must contain the local pieces of the leading *ix+numx-1* by *jx* part of the global matrix.

Scope: **local**

Specified as: an *LLD\_X* by (at least) *LOCq(N\_X)* array, containing numbers of the data type indicated in Table 49 on page 131. Details about the block-cyclic data distribution of the global matrix *X* are stored in *desc\_x*.

*ix* has the following meaning:

If  $incx = M\_X$ , it indicates which row of global matrix  $X$  is used for vector  $x$ .

If  $incx = 1$  and  $incx \neq M\_X$ , it is the row index of global matrix  $X$ , identifying the first element of vector  $x$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ix \leq M\_X$ , and if  $incx = 1$  and  $incx \neq M\_X$ , then:

If  $transa = 'N'$ , then  $ix+n-1 \leq M\_X$ .

If  $transa = 'T'$  or  $'C'$ , then  $ix+m-1 \leq M\_X$ .

$jx$  has the following meaning:

If  $incx = M\_X$ , it is the column index of global matrix  $X$ , identifying the first element of vector  $x$ .

If  $incx = 1$  and  $incx \neq M\_X$ , it indicates which column of global matrix  $X$  is used for vector  $x$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq jx \leq N\_X$ , and if  $incx = M\_X$ , then:

If  $transa = 'N'$ , then  $jx+n-1 \leq N\_X$ .

If  $transa = 'T'$  or  $'C'$ , then  $jx+m-1 \leq N\_X$ .

$desc\_x$  is the array descriptor for global matrix  $X$ , described in the following table:

$desc\_x$	Name	Description	Limits	Scope
1	DTYPE_X	Descriptor type	DTYPE_X=1	Global
2	CTXT_X	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_X	Number of rows in the global matrix	If $transa = 'N'$ and $n = 0$ : $M\_X \geq 0$ If $transa = 'T'$ and $m = 0$ : $M\_X \geq 0$ Otherwise: $M\_X \geq 1$	Global
4	N_X	Number of columns in the global matrix	If $transa = 'N'$ and $n = 0$ : $N\_X \geq 0$ If $transa = 'T'$ and $m = 0$ : $N\_X \geq 0$ Otherwise: $N\_X \geq 1$	Global
5	MB_X	Row block size	$MB\_X \geq 1$	Global
6	NB_X	Column block size	$NB\_X \geq 1$	Global
7	RSRC_X	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_X < p$	Global
8	CSRC_X	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_X < q$	Global
9	LLD_X	The leading dimension of the local array	$LLD\_X \geq \max(1, LOCp(M\_X))$	<b>Local</b>



Specified as: an array of (at least) length 9, containing fullword integers.

*incx* is the stride for global vector  $x$ .

Scope: **global**

Specified as: a fullword integer;  $incx = 1$  or  $incx = M_X$ , where:

If  $incx = M_X$ , then  $x$  is a row-distributed vector.

If  $incx = 1$  and  $incx \neq M_X$ , then  $x$  is a column-distributed vector.

*beta* is the scalar  $\beta$ .

Scope: **global**

Specified as: a number of the data type indicated in Table 49 on page 131.

*y* is the local part of the global matrix  $Y$ . This identifies the **first element** of the local array  $Y$ . This subroutine computes the location of the first element of the local subarray used, based on *iy*, *jy*, *desc\_y*, *p*, *q*, *myrow*, and *mycol*; therefore, assuming the following:

If *transa* = 'N', *numy* = *m*

If *transa* = 'T' or 'C', *numy* = *n*

the following must be true:

- If  $incy = M_Y$ , the leading  $LOCp(iy)$  by  $LOCq(jy+numy-1)$  part of the local array  $Y$  must contain the local pieces of the leading  $iy$  by  $jy+numy-1$  part of the global matrix.
- If  $incy = 1$  and  $incy \neq M_Y$ , the leading  $LOCp(iy+numy-1)$  by  $LOCq(jy)$  part of the local array  $Y$  must contain the local pieces of the leading  $iy+numy-1$  by  $jy$  part of the global matrix.

When  $\beta$  is zero, *y* need not be set on input.

Scope: **local**

Specified as: an  $LLD_Y$  by (at least)  $LOCq(N_Y)$  array, containing numbers of the data type indicated in Table 49 on page 131. Details about the block-cyclic data distribution of the global matrix  $Y$  are stored in *desc\_y*.

*iy* has the following meaning:

If  $incy = M_Y$ , it indicates which row of global matrix  $Y$  is used for vector  $y$ .

If  $incy = 1$  and  $incy \neq M_Y$ , it is the row index of global matrix  $Y$ , identifying the first element of vector  $y$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq iy \leq M_Y$ , and if  $incy = 1$  and  $incy \neq M_Y$ , then:

If *transa* = 'N', then  $iy+m-1 \leq M_Y$ .

If *transa* = 'T' or 'C', then  $iy+n-1 \leq M_Y$ .

*jy* has the following meaning:

If  $incy = M_Y$ , it is the column index of global matrix  $Y$ , identifying the first element of vector  $y$ .

If  $incy = 1$  and  $incy \neq M\_Y$ , it indicates which column of global matrix  $Y$  is used for vector  $y$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq jy \leq N\_Y$ , and if  $incy = M\_Y$ , then:

If  $transa = 'N'$ , then  $jy+m-1 \leq N\_Y$ .

If  $transa = 'T'$  or  $'C'$ , then  $jy+n-1 \leq N\_Y$ .

$desc\_y$  is the array descriptor for global matrix  $Y$ , described in the following table:

$desc\_y$	Name	Description	Limits	Scope
1	DTYPE_Y	Descriptor type	DTYPE_Y=1	Global
2	CTXT_Y	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_Y	Number of rows in the global matrix	If $transa = 'N'$ and $m = 0$ : $M\_Y \geq 0$ If $transa = 'T'$ and $n = 0$ : $M\_Y \geq 0$ Otherwise: $M\_Y \geq 1$	Global
4	N_Y	Number of columns in the global matrix	If $transa = 'N'$ and $m = 0$ : $N\_Y \geq 0$ If $transa = 'T'$ and $n = 0$ : $N\_Y \geq 0$ Otherwise: $N\_Y \geq 1$	Global
5	MB_Y	Row block size	$MB\_Y \geq 1$	Global
6	NB_Y	Column block size	$NB\_Y \geq 1$	Global
7	RSRC_Y	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_Y < p$	Global
8	CSRC_Y	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_Y < q$	Global
9	LLD_Y	The leading dimension of the local array	$LLD\_Y \geq \max(1, LOCp(M\_Y))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

$incy$  is the stride for global vector  $y$ .

Scope: **global**

Specified as: a fullword integer;  $incy = 1$  or  $incy = M\_Y$ , where:

If  $incy = M\_Y$ , then  $y$  is a row-distributed vector.

If  $incy = 1$  and  $incy \neq M\_Y$ , then  $y$  is a column-distributed vector.

## On Return

$y$  is the updated local part of the global matrix  $Y$ , containing the results of the computation.

Scope: **local**

Returned as: an LLD\_Y by (at least) LOCq(N\_Y) array, containing numbers of the data type indicated in Table 49 on page 131.

## Notes and Coding Rules

1. These subroutines accept lowercase letters for the *transa* argument.
2. For PDGEMV, if you specify 'C' for *transa*, it is interpreted as though you specified 'T'.
3. The matrix and vectors must have no common elements; otherwise, results are unpredictable.
4. The NUMROC utility subroutine can be used to determine the values of LOCp(M\_) and LOCq(N\_) used in the argument descriptions above. For details, see "Determining the Number of Rows and Columns in Your Local Arrays" on page 28 and "NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process" on page 868.
5. For suggested block sizes, see "Coding Tips for Optimizing Parallel Performance" on page 77.
6. The following values must be equal: CTXT\_A = CTXT\_X = CTXT\_Y.
7. The following coding rules depend upon the values specified for *transa* and *incx*:
  - If *transa* = 'N' and *incx* = M\_X:
    - The following block sizes must be equal: NB\_A = NB\_X.
    - In the process grid, the process column containing the first column of the submatrix X must also contain the first column of the submatrix A; that is,  $iacol = ixcol$ , where:
      - $iacol = \text{mod}(((ja-1)/NB\_A)+CSRC\_A), q)$
      - $ixcol = \text{mod}(((jx-1)/NB\_X)+CSRC\_X), q)$
    - The block column offset of *x* must be equal to the block column offset of A; that is,  $\text{mod}(jx-1, NB\_X) = \text{mod}(ja-1, NB\_A)$ .
  - If *transa* = 'N' and *incx* = 1( ≠ M\_X):
    - The following block sizes must be equal: NB\_A = MB\_X.
    - The block row offset of *x* must be equal to the block column offset of A; that is,  $\text{mod}(ix-1, MB\_X) = \text{mod}(ja-1, NB\_A)$ .
  - If *transa* = 'T' or 'C' and *incx* = M\_X:
    - The following block sizes must be equal: MB\_A = NB\_X.
    - The block column offset of *x* must be equal to the block row offset of A; that is,  $\text{mod}(jx-1, NB\_X) = \text{mod}(ia-1, MB\_A)$ .
  - If *transa* = 'T' or 'C' and *incx* = 1( ≠ M\_X):
    - The following block sizes must be equal: MB\_A = MB\_X.
    - In the process grid, the process row containing the first row of the submatrix X must also contain the first row of the submatrix A; that is,  $iarow = ixrow$ , where:
      - $iarow = \text{mod}(((ia-1)/MB\_A)+RSRC\_A), p)$
      - $ixrow = \text{mod}(((ix-1)/MB\_X)+RSRC\_X), p)$
    - The block row offset of *x* must be equal to the block row offset of A; that is,  $\text{mod}(ix-1, MB\_X) = \text{mod}(ia-1, MB\_A)$ .
8. The following coding rules depend upon the values specified for *transa* and *incy*:
  - If *transa* = 'N' and *incy* = M\_Y:
    - The following block sizes must be equal: MB\_A = NB\_Y.

- The block column offset of  $y$  must be equal to the block row offset of  $A$ ; that is,  $\text{mod}(jy-1, \text{NB\_Y}) = \text{mod}(ia-1, \text{MB\_A})$ .
  - If  $\text{transa} = 'N'$  and  $\text{incy} = 1$  ( $\neq \text{M\_Y}$ ):
    - The following block sizes must be equal:  $\text{MB\_A} = \text{MB\_Y}$ .
    - In the process grid, the process row containing the first row of the submatrix  $Y$  must also contain the first row of the submatrix  $A$ ; that is,  $i\text{arow} = i\text{yrow}$ , where:
      - $i\text{arow} = \text{mod}((((ia-1)/\text{MB\_A})+\text{RSRC\_A}), p)$
      - $i\text{yrow} = \text{mod}((((iy-1)/\text{MB\_Y})+\text{RSRC\_Y}), p)$
    - The block row offset of  $y$  must be equal to the block row offset of  $A$ ; that is,  $\text{mod}(iy-1, \text{MB\_Y}) = \text{mod}(ia-1, \text{MB\_A})$ .
  - If  $\text{transa} = 'T'$  or  $'C'$  and  $\text{incy} = \text{M\_Y}$ :
    - The following block sizes must be equal:  $\text{NB\_A} = \text{NB\_Y}$ .
    - In the process grid, the process column containing the first column of the submatrix  $Y$  must also contain the first column of the submatrix  $A$ ; that is,  $i\text{acol} = i\text{ycol}$ , where:
      - $i\text{acol} = \text{mod}((((ja-1)/\text{NB\_A})+\text{CSRC\_A}), q)$
      - $i\text{ycol} = \text{mod}((((jy-1)/\text{NB\_Y})+\text{CSRC\_Y}), q)$
    - The block column offset of  $y$  must be equal to the block column offset of  $A$ ; that is,  $\text{mod}(jy-1, \text{NB\_Y}) = \text{mod}(ja-1, \text{NB\_A})$ .
  - If  $\text{transa} = 'T'$  or  $'C'$  and  $\text{incy} = 1$  ( $\neq \text{M\_Y}$ ):
    - The following block sizes must be equal:  $\text{NB\_A} = \text{MB\_Y}$ .
    - The block row offset of  $y$  must be equal to the block column offset of  $A$ ; that is,  $\text{mod}(iy-1, \text{MB\_Y}) = \text{mod}(ja-1, \text{NB\_A})$ .
9. An example of the use of this subroutine in a thermal diffusion application program is shown in Appendix B, “Sample Programs.” See “Program Main” on page 902.

## Error Conditions

### Computational Errors

None

### Resource Errors

Unable to allocate work space

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1.  $\text{DTYPE\_A}$  is invalid.
2.  $\text{DTYPE\_X}$  is invalid.
3.  $\text{DTYPE\_Y}$  is invalid.

#### Stage 2:

1.  $\text{CTXT\_A}$  is invalid.

#### Stage 3:

1. This subroutine was called from outside the process grid.

#### Stage 4:

1.  $\text{transa} \neq 'N', 'T', \text{ or } 'C'$
2.  $m < 0$
3.  $n < 0$

4.  $M\_A < 0$  and  $(m = 0 \text{ or } n = 0)$ ;  $M\_A < 1$  otherwise
5.  $N\_A < 0$  and  $(m = 0 \text{ or } n = 0)$ ;  $N\_A < 1$  otherwise
6.  $MB\_A < 1$
7.  $NB\_A < 1$
8.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
9.  $CSRC\_A < 0$  or  $CSRC\_A \geq q$
10.  $ia < 1$
11.  $ja < 1$ 
  - If  $(n = 0 \text{ and } transa = 'N')$  or  $(m = 0 \text{ and } transa = 'T' \text{ or } 'C')$ :
12.  $M\_X < 0$
13.  $N\_X < 0$ 
  - Otherwise:
14.  $M\_X < 1$
15.  $N\_X < 1$ 
  - In all cases:
16.  $MB\_X < 1$
17.  $NB\_X < 1$
18.  $RSRC\_X < 0$  or  $RSRC\_X \geq p$
19.  $CSRC\_X < 0$  or  $CSRC\_X \geq q$
20.  $CTXT\_A \neq CTXT\_X$
21.  $ix < 1$
22.  $jx < 1$ 
  - If  $(m = 0 \text{ and } transa = 'N')$  or  $(n = 0 \text{ and } transa = 'T' \text{ or } 'C')$ :
23.  $M\_Y < 0$
24.  $N\_Y < 0$ 
  - Otherwise:
25.  $M\_Y < 1$
26.  $N\_Y < 1$ 
  - In all cases:
27.  $MB\_Y < 1$
28.  $NB\_Y < 1$
29.  $RSRC\_Y < 0$  or  $RSRC\_Y \geq p$
30.  $CSRC\_Y < 0$  or  $CSRC\_Y \geq q$
31.  $CTXT\_A \neq CTXT\_Y$
32.  $iy < 1$
33.  $jy < 1$

**Stage 5:** If  $m \neq 0$  and  $n \neq 0$ :

1.  $ia > M\_A$
2.  $ja > N\_A$
3.  $ia+m-1 > M\_A$
4.  $ja+n-1 > N\_A$ 
  - If  $(n \neq 0 \text{ and } transa = 'N')$  or  $(m \neq 0 \text{ and } transa = 'T' \text{ or } 'C')$ :
5.  $ix > M\_X$
6.  $jx > N\_X$ 
  - If  $(m \neq 0 \text{ and } transa = 'N')$  or  $(n \neq 0 \text{ and } transa = 'T' \text{ or } 'C')$ :
7.  $iy > M\_Y$
8.  $jy > N\_Y$ 
  - If  $incx = M\_X$  and  $transa = 'N'$ :
9.  $NB\_X \neq NB\_A$
10.  $\text{mod}(jx-1, NB\_X) \neq \text{mod}(ja-1, NB\_A)$
11.  $n \neq 0$  and  $jx+n-1 \leq N\_X$ 
  - If  $incx = M\_X$  and  $transa = 'T' \text{ or } 'C'$ :
12.  $NB\_X \neq MB\_A$

13.  $\text{mod}(jx-1, \text{NB\_X}) \neq \text{mod}(ia-1, \text{MB\_A})$
14.  $m \neq 0$  and  $jx+m-1 \leq \text{N\_X}$   
If  $\text{incx} = 1$  ( $\neq \text{M\_X}$ ) and  $\text{transa} = 'N'$ :
15.  $\text{MB\_X} \neq \text{NB\_A}$
16.  $\text{mod}(ix-1, \text{MB\_X}) \neq \text{mod}(ja-1, \text{NB\_A})$
17.  $n \neq 0$  and  $ix+n-1 \leq \text{M\_X}$   
If  $\text{incx} = 1$  ( $\neq \text{M\_X}$ ) and  $\text{transa} = 'T'$  or  $'C'$ :
18.  $\text{MB\_X} \neq \text{MB\_A}$
19.  $\text{mod}(ix-1, \text{MB\_X}) \neq \text{mod}(ia-1, \text{MB\_A})$
20.  $m \neq 0$  and  $ix+m-1 \leq \text{M\_X}$   
In all cases:
21.  $\text{incx} \neq \text{M\_X}$  and  $\text{incx} \neq 1$   
If  $\text{incy} = \text{M\_Y}$  and  $\text{transa} = 'N'$ :
22.  $\text{NB\_Y} \neq \text{MB\_A}$
23.  $\text{mod}(jy-1, \text{NB\_Y}) \neq \text{mod}(ia-1, \text{MB\_A})$
24.  $m \neq 0$  and  $jy+m-1 \leq \text{N\_Y}$   
If  $\text{incy} = \text{M\_Y}$  and  $\text{transa} = 'T'$  or  $'C'$ :
25.  $\text{NB\_Y} \neq \text{NB\_A}$
26.  $\text{mod}(jy-1, \text{NB\_Y}) \neq \text{mod}(ja-1, \text{NB\_A})$
27.  $n \neq 0$  and  $jy+n-1 \leq \text{N\_Y}$   
If  $\text{incy} = 1$  ( $\neq \text{M\_Y}$ ) and  $\text{transa} = 'N'$ :
28.  $\text{MB\_Y} \neq \text{MB\_A}$
29.  $\text{mod}(iy-1, \text{MB\_Y}) \neq \text{mod}(ia-1, \text{MB\_A})$
30.  $m \neq 0$  and  $iy+m-1 \leq \text{M\_Y}$   
If  $\text{incy} = 1$  ( $\neq \text{M\_Y}$ ) and  $\text{transa} = 'T'$  or  $'C'$ :
31.  $\text{MB\_Y} \neq \text{NB\_A}$
32.  $\text{mod}(iy-1, \text{MB\_Y}) \neq \text{mod}(ja-1, \text{NB\_A})$
33.  $n \neq 0$  and  $iy+n-1 \leq \text{M\_Y}$   
In all cases:
34.  $\text{incy} \neq \text{M\_Y}$  and  $\text{incy} \neq 1$

**Stage 6:** If  $\text{transa} = 'N'$ :

1. If  $\text{incx} = \text{M\_X}$ , then (in the process grid) the process column containing the first column of the submatrix  $X$  does not contain the first column of the submatrix  $A$ ; that is,  $\text{iacol} \neq \text{ixcol}$ , where:
  - $\text{iacol} = \text{mod}(\text{mod}((ja-1)/\text{NB\_A}) + \text{CSRC\_A}, q)$
  - $\text{ixcol} = \text{mod}(\text{mod}((jx-1)/\text{NB\_X}) + \text{CSRC\_X}, q)$
2. If  $\text{incy} = 1$  ( $\neq \text{M\_Y}$ ), then (in the process grid) the process row containing the first row of the submatrix  $Y$  does not contain the first row of the submatrix  $A$ ; that is,  $\text{iarow} \neq \text{iyrow}$ , where:
  - $\text{iarow} = \text{mod}(\text{mod}((ia-1)/\text{MB\_A}) + \text{RSRC\_A}, p)$
  - $\text{iyrow} = \text{mod}(\text{mod}((iy-1)/\text{MB\_Y}) + \text{RSRC\_Y}, p)$
 If  $\text{transa} = 'T'$  or  $'C'$ :
3. If  $\text{incx} = 1$  ( $\neq \text{M\_X}$ ), then (in the process grid) the process row containing the first row of the submatrix  $X$  does not contain the first row of the submatrix  $A$ ; that is,  $\text{iarow} \neq \text{ixrow}$ , where:
  - $\text{iarow} = \text{mod}(\text{mod}((ia-1)/\text{MB\_A}) + \text{RSRC\_A}, p)$
  - $\text{ixrow} = \text{mod}(\text{mod}((ix-1)/\text{MB\_X}) + \text{RSRC\_X}, p)$
4. If  $\text{incy} = \text{M\_Y}$ , then (in the process grid) the process column containing the first column of the submatrix  $Y$  does not contain the first column of the submatrix  $A$ ; that is,  $\text{iacol} \neq \text{iycol}$ , where:
  - $\text{iacol} = \text{mod}(\text{mod}((ja-1)/\text{NB\_A}) + \text{CSRC\_A}, q)$
  - $\text{iycol} = \text{mod}(\text{mod}((jy-1)/\text{NB\_Y}) + \text{CSRC\_Y}, q)$

In all cases:

5.  $LLD\_A < \max(1, LOCp(M\_A))$
6.  $LLD\_X < \max(1, LOCp(M\_X))$
7.  $LLD\_Y < \max(1, LOCp(M\_Y))$

## Examples

### Example 1

This example computes  $y = \alpha Ax + \beta y$  using a  $2 \times 2$  process grid. The input matrices  $A$ ,  $X$ , and  $Y$ , used here, are the same as  $A$ ,  $B$ , and  $C$ , used in “Example 1” on page 250 for PDGEMM. The updated portion of  $Y$  is the same as for  $C$  in PDGEMM, as this computation is equivalent to a portion of the PDGEMM computation.

This example uses a global submatrix  $A$  within a global matrix  $A$  by specifying  $ia = 3$  and  $ja = 1$ . It uses vectors  $x$  and  $y$ , which are column-distributed vectors within a column of  $X$  and  $Y$ , respectively, by specifying  $incx = 1$ ,  $ix = 1$ , and  $jx = 2$  for  $x$  and  $incy = 1$ ,  $iy = 3$ , and  $jy = 2$  for  $y$ .

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      TRANSA M   N   ALPHA   A   IA   JA   DESC_A   X   IX   JX
      |      |   |       |   |   |   |         |   |   |
CALL PDGEMV( 'N' , 4 , 5 , 1.0D0 , A , 3 , 1 , DESC_A , X , 1 , 2 ,

      DESC_X INCX BETA   Y   IY   JY   DESC_Y INCY
      |      |   |       |   |   |   |         |
      DESC_X , 1 , 2.0D0 , Y , 3 , 2 , DESC_Y , 1 )
```

	Desc_A	Desc_X	Desc_Y
DTYPE_	1	1	1
CTXT_	$icontxt^1$	$icontxt^1$	$icontxt^1$
M_	6	5	6
N_	5	4	4
MB_	3	2	3
NB_	2	2	2
RSRC_	0	0	0
CSRC_	0	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>	See below <sup>2</sup>

#### Notes:

1.  $icontxt$  is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:

```
LLD_A = MAX(1, NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_X = MAX(1, NUMROC(M_X, MB_X, MYROW, RSRC_X, NPROW))
LLD_Y = MAX(1, NUMROC(M_Y, MB_Y, MYROW, RSRC_Y, NPROW))
```

In this example,  $LLD\_A = LLD\_Y = 3$  on all processes,  $LLD\_X = 3$  on  $P_{00}$  and  $P_{01}$ , and  $LLD\_X = 2$  on  $P_{10}$  and  $P_{11}$ .

## PDGEMV and PZGEMV

After the global matrix  $A$  is distributed over the process grid, only a portion of the global data structure is used—that is, global submatrix  $A$ . Following is the global  $4 \times 5$  submatrix  $A$ , starting at row 3 and column 1 in global general  $6 \times 5$  matrix  $A$  with block size  $3 \times 2$ :

B,D	0	1	2
0	$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \\ 1.0 & -1.0 \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \\ -1.0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} \cdot \\ \cdot \\ 2.0 \end{bmatrix}$
1	$\begin{bmatrix} -3.0 & 2.0 \\ 4.0 & 0.0 \\ -1.0 & -1.0 \end{bmatrix}$	$\begin{bmatrix} 2.0 & 2.0 \\ -2.0 & 1.0 \\ 1.0 & -3.0 \end{bmatrix}$	$\begin{bmatrix} 0.0 \\ -1.0 \\ 2.0 \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$

Local arrays for  $A$ :

p,q	0	1
0	$\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ 1.0 & -1.0 & 2.0 \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \\ -1.0 & 1.0 \end{bmatrix}$
1	$\begin{bmatrix} -3.0 & 2.0 & 0.0 \\ 4.0 & 0.0 & -1.0 \\ -1.0 & -1.0 & 2.0 \end{bmatrix}$	$\begin{bmatrix} 2.0 & 2.0 \\ -2.0 & 1.0 \\ 1.0 & -3.0 \end{bmatrix}$

After the global matrix  $X$  is distributed over the process grid, only a portion of the global data structure is used—that is, global vector  $x$ , which is a column-distributed vector. Following is the global vector  $x$  of size  $5 \times 1$ , starting at row 1 and column 2 in  $5 \times 4$  global matrix  $X$  with block size  $2 \times 2$ :

B,D	0	1
0	$\begin{bmatrix} \cdot & -1.0 \\ \cdot & 2.0 \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}$
1	$\begin{bmatrix} \cdot & 0.0 \\ \cdot & -1.0 \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}$
2	$\begin{bmatrix} \cdot & 2.0 \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0	1
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

Local arrays for  $x$ :

p,q	0	1
0	$\begin{bmatrix} \cdot & -1.0 \\ \cdot & 2.0 \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}$



			.	2.0	
1			.	0.0	
			.	-1.0	

After the global matrix  $Y$  is distributed over the process grid, only a portion of the global data structure is used—that is, global vector  $y$ , which is a column-distributed vector. Following is the global vector  $y$  of size  $4 \times 1$ , starting at row 3 and column 2 in  $6 \times 4$  global matrix  $Y$  with block size  $3 \times 2$ :

B,D					
				0	1
0			.	.	
			.	.	
			.	0.5	
1			.	0.5	
			.	0.5	
			.	0.5	

The following is the  $2 \times 2$  process grid:

B,D					
				0	1
0				$P_{00}$	$P_{01}$
1				$P_{10}$	$P_{11}$

Local arrays for  $y$ :

p,q					
				0	1
0			.	.	
			.	.	
			.	0.5	
1			.	0.5	
			.	0.5	
			.	0.5	

### Output:

After the global matrix  $Y$  is distributed over the process grid, only a portion of the global data structure is used—that is, global vector  $y$ , which is a column-distributed vector. Following is the global vector  $y$  of size  $4 \times 1$ , starting at row 3 and column 2 in  $6 \times 4$  global matrix  $Y$  with block size  $3 \times 2$ :

B,D					
				0	1
0			.	.	
			.	.	
			.	1.0	
1			.	6.0	
			.	-6.0	
			.	7.0	

The following is the  $2 \times 2$  process grid:

B,D					
				0	1
0				$P_{00}$	$P_{01}$
1				$P_{10}$	$P_{11}$

Local arrays for  $y$ :

p,q	0	1
0	$\begin{matrix} \cdot & \cdot \\ \cdot & \cdot \\ \cdot & 1.0 \end{matrix}$	$\begin{matrix} \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \end{matrix}$
1	$\begin{matrix} \cdot & 6.0 \\ \cdot & -6.0 \\ \cdot & 7.0 \end{matrix}$	$\begin{matrix} \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \end{matrix}$

## Example 2

This example computes  $y = \alpha Ax + \beta y$  using a  $2 \times 2$  process grid. The input matrices  $A$ ,  $X$ , and  $Y$ , used here, are the same as  $A$ ,  $B$ , and  $C$ , used in “Example 1” on page 250 for PDGEMM.

This example uses a global submatrix  $A$  within a global matrix  $A$  by specifying  $ia = 2$  and  $ja = 2$ . It uses vector  $x$ , which is a row-distributed vector within a row of  $X$ , by specifying  $incx = M\_X = 5$ ,  $ix = 4$ , and  $jx = 2$ . It uses vector  $y$ , which is a column-distributed vector within a column of  $Y$ , by specifying  $incy = 1$ ,  $iy = 2$ , and  $jy = 3$ .

### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      TRANSA M   N   ALPHA   A   IA   JA   DESC_A   X   IX   JX
      |      |   |   |      |   |   |   |      |   |   |
CALL PDGEMV( 'N' , 4 , 3 , 1.0D0 , A , 2 , 2 , DESC_A , X , 4 , 2 ,

      DESC_X INCX   BETA   Y   IY   JY   DESC_Y   INCY
      |      |   |   |   |   |   |   |   |
      DESC_X , 5 , 2.0D0 , Y , 2 , 3 , DESC_Y , 1 )
```

	Desc_A	Desc_X	Desc_Y
DTYPE_	1	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	6	5	6
N_	5	4	4
MB_	3	2	3
NB_	2	2	2
RSRC_	0	0	0
CSRC_	0	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>	See below <sup>2</sup>

### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:

```
LLD_A = MAX(1,NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_X = MAX(1,NUMROC(M_X, MB_X, MYROW, RSRC_X, NPROW))
LLD_Y = MAX(1,NUMROC(M_Y, MB_Y, MYROW, RSRC_Y, NPROW))
```

In this example,  $LLD\_A = LLD\_Y = 3$  on all processes,  $LLD\_X = 3$  on  $P_{00}$  and  $P_{01}$ , and  $LLD\_X = 2$  on  $P_{10}$  and  $P_{11}$ .

After the global matrix  $A$  is distributed over the process grid, only a portion of the global data structure is used—that is, global submatrix  $A$ . Following is the global  $4 \times 3$  submatrix  $A$ , starting at row 2 and column 2 in global general  $6 \times 5$  matrix  $A$  with block size  $3 \times 2$ :

B,D	0	1	2
0	$\begin{bmatrix} \cdot & \cdot \\ \cdot & 0.0 \\ \cdot & -1.0 \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot \\ 1.0 & 1.0 \\ -1.0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix}$
1	$\begin{bmatrix} \cdot & 2.0 \\ \cdot & 0.0 \\ \cdot & \cdot \end{bmatrix}$	$\begin{bmatrix} 2.0 & 2.0 \\ -2.0 & 1.0 \\ \cdot & \cdot \end{bmatrix}$	$\begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$

Local arrays for  $A$ :

p,q	0	1
0	$\begin{bmatrix} \cdot & \cdot \\ \cdot & 0.0 \\ \cdot & -1.0 \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot \\ 1.0 & 1.0 \\ -1.0 & 1.0 \end{bmatrix}$
1	$\begin{bmatrix} \cdot & 2.0 \\ \cdot & 0.0 \\ \cdot & \cdot \end{bmatrix}$	$\begin{bmatrix} 2.0 & 2.0 \\ -2.0 & 1.0 \\ \cdot & \cdot \end{bmatrix}$

After the global matrix  $X$  is distributed over the process grid, only a portion of the global data structure is used—that is, global vector  $x$ , which is a row-distributed vector. Following is the global vector  $x$  of size  $1 \times 3$ , starting at row 4 and column 2 in  $5 \times 4$  global matrix  $X$  with block size  $2 \times 2$ :

B,D	0	1
0	$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}$
1	$\begin{bmatrix} \cdot & \cdot \\ \cdot & -1.0 \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot \\ 1.0 & -1.0 \end{bmatrix}$
2	$\begin{bmatrix} \cdot & \cdot \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0	1
0	$P_{00}$	$P_{01}$
2	$P_{10}$	$P_{11}$

Local arrays for  $x$ :

## PDGEMV and PZGEMV

p,q	0	1
0	$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}$
1	$\begin{bmatrix} \cdot & \cdot \\ \cdot & -1.0 \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot \\ 1.0 & -1.0 \end{bmatrix}$

After the global matrix  $Y$  is distributed over the process grid, only a portion of the global data structure is used—that is, global vector  $y$ , which is a column-distributed vector. Following is the global vector  $y$  of size  $4 \times 1$ , starting at row 2 and column 3 in  $6 \times 4$  global matrix  $Y$  with block size  $3 \times 2$ :

B,D	0	1
0	$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot \\ 0.5 & \cdot \\ 0.5 & \cdot \end{bmatrix}$
1	$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot \\ 0.5 & \cdot \\ 0.5 & \cdot \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0	1
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$

Local arrays for  $y$ :

p,q	0	1
0	$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot \\ 0.5 & \cdot \\ 0.5 & \cdot \end{bmatrix}$
1	$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot \\ 0.5 & \cdot \\ 0.5 & \cdot \end{bmatrix}$

### Output:

After the global matrix  $Y$  is distributed over the process grid, only a portion of the global data structure is used—that is, global vector  $y$ , which is a column-distributed vector. Following is the global vector  $y$  of size  $4 \times 1$ , starting at row 2 and column 3 in  $6 \times 4$  global matrix  $Y$  with block size  $3 \times 2$ :

B,D	0	1
0	$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot \\ 1.0 & \cdot \\ 0.0 & \cdot \end{bmatrix}$
1	$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot \\ -1.0 & \cdot \\ -2.0 & \cdot \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0	1
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$

Local arrays for  $y$ :

p,q	0	1
0	$\begin{matrix} \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \end{matrix}$	$\begin{matrix} \cdot & \cdot \\ 1.0 & \cdot \\ 0.0 & \cdot \end{matrix}$
1	$\begin{matrix} \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \end{matrix}$	$\begin{matrix} -1.0 & \cdot \\ -2.0 & \cdot \\ \cdot & \cdot \end{matrix}$

### Example 3

This example computes  $y = \alpha Ax + \beta y$  using a  $2 \times 2$  process grid. The input matrices  $A$ ,  $X$ , and  $Y$ , used here, are the same as  $A$ ,  $B$ , and  $C$ , used in “Example 2” on page 252 for PZGEMM. The updated portion of  $Y$  is the same as for  $C$  in PZGEMM, as this computation is equivalent to a portion of the PZGEMM computation.

This example uses vectors  $x$  and  $y$ , which are column-distributed vectors within a column of  $X$  and  $Y$ , respectively, by specifying  $incx = 1$ ,  $ix = 1$ , and  $jx = 2$  for  $x$  and  $incy = 1$ ,  $iy = 1$ , and  $jy = 2$  for  $y$ .

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      TRANSA M   N   ALPHA   A   IA   JA   DESC_A   X   IX   JX
      |      |   |   |      |   |   |   |      |   |   |
CALL PZGEMV( 'N' , 6 , 3 , ALPHA , A , 1 , 1 , DESC_A , X , 1 , 2 ,

      DESC_X INCX BETA   Y   IY   JY   DESC_Y   INCY
      |      |   |   |   |   |   |      |   |
      DESC_X , 1 , BETA , Y , 1 , 2 , DESC_Y , 1 )

ALPHA = (1.0,0.0)

BETA  = (2.0,0.0)
```

	Desc_A	Desc_X	Desc_Y
DTYPE_	1	1	1
CTXT_	$icontxt^1$	$icontxt^1$	$icontxt^1$
M_	6	3	6
N_	3	2	2
MB_	2	2	2
NB_	2	2	2
RSRC_	0	0	0
CSRC_	0	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>	See below <sup>2</sup>

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:

```
LLD_A = MAX(1,NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_X = MAX(1,NUMROC(M_X, MB_X, MYROW, RSRC_X, NPROW))
LLD_Y = MAX(1,NUMROC(M_Y, MB_Y, MYROW, RSRC_Y, NPROW))
```

In this example:

```
LLD_A = LLD_Y = 4 on P00 and P01
LLD_X = 2 on P00 and P01
LLD_A = LLD_Y = 2 on P10 and P11
LLD_X = 1 on P10 and P11
```

Global general  $6 \times 3$  matrix *A* with block size  $2 \times 2$ :

B,D	0	1
0	$\begin{bmatrix} (1.0,5.0) & (9.0,2.0) \\ (2.0,4.0) & (8.0,3.0) \end{bmatrix}$	$\begin{bmatrix} (1.0,9.0) \\ (1.0,8.0) \end{bmatrix}$
1	$\begin{bmatrix} (3.0,3.0) & (7.0,5.0) \\ (4.0,2.0) & (4.0,7.0) \end{bmatrix}$	$\begin{bmatrix} (1.0,7.0) \\ (1.0,5.0) \end{bmatrix}$
2	$\begin{bmatrix} (5.0,1.0) & (5.0,1.0) \\ (6.0,6.0) & (3.0,6.0) \end{bmatrix}$	$\begin{bmatrix} (1.0,6.0) \\ (1.0,4.0) \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0	1
0	P <sub>00</sub>	P <sub>01</sub>
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for *A*:

p,q	0	1
0	$\begin{bmatrix} (1.0,5.0) & (9.0,2.0) \\ (2.0,4.0) & (8.0,3.0) \\ (5.0,1.0) & (5.0,1.0) \\ (6.0,6.0) & (3.0,6.0) \end{bmatrix}$	$\begin{bmatrix} (1.0,9.0) \\ (1.0,8.0) \\ (1.0,6.0) \\ (1.0,4.0) \end{bmatrix}$
1	$\begin{bmatrix} (3.0,3.0) & (7.0,5.0) \\ (4.0,2.0) & (4.0,7.0) \end{bmatrix}$	$\begin{bmatrix} (1.0,7.0) \\ (1.0,5.0) \end{bmatrix}$

After the global matrix *X* is distributed over the process grid, only a portion of the global data structure is used—that is, global vector *x*, which is a column-distributed vector. Following is the global vector *x* of size  $3 \times 1$ , starting at row 1 and column 2 in  $3 \times 2$  global matrix *X* with block size  $2 \times 2$ :

B,D	0
0	$\begin{bmatrix} . & (2.0,7.0) \\ . & (6.0,8.0) \end{bmatrix}$
1	$\begin{bmatrix} . & (4.0,5.0) \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0	--
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$

Local arrays for  $x$ :

p,q	0
0	<div> <div>.</div> <div>(2.0,7.0)</div> <div>.</div> <div>(6.0,8.0)</div> </div>
1	<div> <div>.</div> <div>(4.0,5.0)</div> </div>

After the global matrix  $Y$  is distributed over the process grid, only a portion of the global data structure is used—that is, global vector  $y$ , which is a column-distributed vector. Following is the global vector  $y$  of size  $6 \times 1$ , starting at row 1 and column 2 in  $6 \times 2$  global matrix  $Y$  with block size  $2 \times 2$ :

B,D	0
0	<div> <div>.</div> <div>(0.5,0.0)</div> <div>.</div> <div>(0.5,0.0)</div> </div>
1	<div> <div>.</div> <div>(0.5,0.0)</div> <div>.</div> <div>(0.5,0.0)</div> </div>
2	<div> <div>.</div> <div>(0.5,0.0)</div> <div>.</div> <div>(0.5,0.0)</div> </div>

The following is the  $2 \times 2$  process grid:

B,D	0	--
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

Local arrays for  $y$ :

p,q	0
0	<div> <div>.</div> <div>(0.5,0.0)</div> <div>.</div> <div>(0.5,0.0)</div> <div>.</div> <div>(0.5,0.0)</div> <div>.</div> <div>(0.5,0.0)</div> </div>
1	<div> <div>.</div> <div>(0.5,0.0)</div> <div>.</div> <div>(0.5,0.0)</div> </div>

### Output:

After the global matrix  $Y$  is distributed over the process grid, only a portion of the global data structure is used—that is, global vector  $y$ , which is a column-distributed vector. Following is the global vector  $y$  of size  $6 \times 1$ , starting at row 1 and column 2 in  $6 \times 2$  global matrix  $Y$  with block size  $2 \times 2$ :

B,D	0
0	<div> <div>.</div> <div>(-35.0,142.0)</div> <div>.</div> <div>(-35.0,141.0)</div> </div>
1	<div> <div>.</div> <div>(-43.0,146.0)</div> <div>.</div> <div>(-58.0,131.0)</div> </div>

$$2 \left[ \begin{array}{c|c} \text{-----} & \\ \cdot & (0.0, 112.0) \\ \cdot & (-75.0, 135.0) \end{array} \right]$$

The following is the  $2 \times 2$  process grid:

B,D	0	--
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for  $y$ :

p,q	0
	-----
	· (-35.0, 142.0)
	· (-35.0, 141.0)
0	· ( 0.0, 112.0)
	· (-75.0, 135.0)
	-----
1	· (-43.0, 146.0)
	· (-58.0, 131.0)

#### Example 4

This example computes  $y = \alpha Ax + \beta y$  using a  $2 \times 2$  process grid. The input matrices  $A$ ,  $X$ , and  $Y$ , used here, are the same as  $A$ ,  $B$ , and  $C$ , used in “Example 2” on page 252 for PZGEMM.

This example uses vector  $x$ , which is a row-distributed vector within a row of  $X$ , by specifying  $incx = M\_X = 3$ ,  $ix = 1$ , and  $jx = 1$ . It uses vector  $y$ , which is a column-distributed vector within a column of  $Y$ , by specifying  $incy = 1$ ,  $iy = 1$ , and  $jy = 1$ .

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      TRANS M   N   ALPHA   A   IA   JA   DESC_A   X   IX   JX
      |   |   |   |   |   |   |   |   |   |   |
CALL PZGEMV( 'N' , 6 , 2 , ALPHA , A , 1 , 1 , DESC_A , X , 1 , 1 ,

      DESC_X INCX BETA   Y   IY   JY   DESC_Y   INCY
      |   |   |   |   |   |   |   |   |
      DESC_X , 3 , BETA , Y , 1 , 1 , DESC_Y , 1 )

ALPHA = (1.0,0.0)

BETA  = (2.0,0.0)
```

	Desc_A	Desc_X	Desc_Y
DTYPE_	1	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	6	3	6
N_	3	2	2



	Desc_A	Desc_X	Desc_Y
MB_	2	2	2
NB_	2	2	2
RSRC_	0	0	0
CSRC_	0	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>	See below <sup>2</sup>

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:

```

LLD_A = MAX(1, NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_X = MAX(1, NUMROC(M_X, MB_X, MYROW, RSRC_X, NPROW))
LLD_Y = MAX(1, NUMROC(M_Y, MB_Y, MYROW, RSRC_Y, NPROW))

```

In this example:

```

LLD_A = LLD_Y = 4 on P00 and P01
LLD_X = 2 on P00 and P01
LLD_A = LLD_Y = 2 on P10 and P11
LLD_X = 1 on P10 and P11

```

Global general  $6 \times 3$  matrix  $A$  with block size  $2 \times 2$ :

B,D	0	1
0	$\begin{bmatrix} (1.0, 5.0) & (9.0, 2.0) \\ (2.0, 4.0) & (8.0, 3.0) \end{bmatrix}$	$\begin{bmatrix} (1.0, 9.0) \\ (1.0, 8.0) \end{bmatrix}$
1	$\begin{bmatrix} (3.0, 3.0) & (7.0, 5.0) \\ (4.0, 2.0) & (4.0, 7.0) \end{bmatrix}$	$\begin{bmatrix} (1.0, 7.0) \\ (1.0, 5.0) \end{bmatrix}$
2	$\begin{bmatrix} (5.0, 1.0) & (5.0, 1.0) \\ (6.0, 6.0) & (3.0, 6.0) \end{bmatrix}$	$\begin{bmatrix} (1.0, 6.0) \\ (1.0, 4.0) \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0	1
0	P <sub>00</sub>	P <sub>01</sub>
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for  $A$ :

p,q	0	1
0	$\begin{bmatrix} (1.0, 5.0) & (9.0, 2.0) \\ (2.0, 4.0) & (8.0, 3.0) \\ (5.0, 1.0) & (5.0, 1.0) \\ (6.0, 6.0) & (3.0, 6.0) \end{bmatrix}$	$\begin{bmatrix} (1.0, 9.0) \\ (1.0, 8.0) \\ (1.0, 6.0) \\ (1.0, 4.0) \end{bmatrix}$
1	$\begin{bmatrix} (3.0, 3.0) & (7.0, 5.0) \\ (4.0, 2.0) & (4.0, 7.0) \end{bmatrix}$	$\begin{bmatrix} (1.0, 7.0) \\ (1.0, 5.0) \end{bmatrix}$

After the global matrix  $X$  is distributed over the process grid, only a portion of the global data structure is used—that is, global vector  $x$ , which is a row-distributed vector. Following is the global vector  $x$  of size  $1 \times 2$ , starting at row 1 and column 1 in  $3 \times 2$  global matrix  $X$  with block size  $2 \times 2$ :

## PDGEMV and PZGEMV

B,D	0
0	(1.0,8.0) (2.0,7.0)
	.
1	.

The following is the  $2 \times 2$  process grid:

B,D	0	--
0	P <sub>00</sub>	P <sub>01</sub>
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for  $x$ :

p,q	0
0	(1.0,8.0) (2.0,7.0)
	.
1	.

After the global matrix  $Y$  is distributed over the process grid, only a portion of the global data structure is used—that is, global vector  $y$ , which is a column-distributed vector. Following is the global vector  $y$  of size  $6 \times 1$ , starting at row 1 and column 1 in  $6 \times 2$  global matrix  $Y$  with block size  $2 \times 2$ :

B,D	0
0	(0.5,0.0) .
	(0.5,0.0) .
1	(0.5,0.0) .
	(0.5,0.0) .
2	(0.5,0.0) .
	(0.5,0.0) .

The following is the  $2 \times 2$  process grid:

B,D	0	--
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for  $y$ :

p,q	0
	(0.5,0.0) .
	(0.5,0.0) .
0	(0.5,0.0) .
	(0.5,0.0) .
1	(0.5,0.0) .
	(0.5,0.0) .

### Output:

After the global matrix  $Y$  is distributed over the process grid, only a portion of the global data structure is used—that is, global vector  $y$ , which is a

column-distributed vector. Following is the global vector  $y$  of size  $6 \times 1$ , starting at row 1 and column 1 in  $6 \times 2$  global matrix  $Y$  with block size  $2 \times 2$ :

B,D	0	
0	$\begin{pmatrix} -34.0, & 80.0 \\ -34.0, & 82.0 \end{pmatrix}$	$\begin{pmatrix} . \\ . \end{pmatrix}$
1	$\begin{pmatrix} -41.0, & 86.0 \\ -52.0, & 76.0 \end{pmatrix}$	$\begin{pmatrix} . \\ . \end{pmatrix}$
2	$\begin{pmatrix} 1.0, & 78.0 \\ -77.0, & 87.0 \end{pmatrix}$	$\begin{pmatrix} . \\ . \end{pmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0	--
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$

Local arrays for  $y$ :

p,q	0	
0	$\begin{pmatrix} -34.0, & 80.0 \\ -34.0, & 82.0 \\ 1.0, & 78.0 \\ -77.0, & 87.0 \end{pmatrix}$	$\begin{pmatrix} . \\ . \\ . \\ . \end{pmatrix}$
1	$\begin{pmatrix} -41.0, & 86.0 \\ -52.0, & 76.0 \end{pmatrix}$	$\begin{pmatrix} . \\ . \end{pmatrix}$

## PDSYMV and PZHEMV — Matrix-Vector Product for a Real Symmetric or a Complex Hermitian Matrix

### Purpose

These subroutines compute the matrix-vector product:

- $y \leftarrow \alpha Ax + \beta y$

where, in the formula above:

- $A$  represents the global submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$ .
- $x$  represents the global vector:
  - For  $incx = M\_X$ , it is  $X_{ix:ix, jx:jx+n-1}$ .
  - For  $incx = 1$  and  $incx \neq M\_X$ , it is  $X_{ix:ix+n-1, jx:jx}$ .
- $y$  represents the global vector:
  - For  $incy = M\_Y$ , it is  $Y_{iy:iy, jy:jy+n-1}$ .
  - For  $incy = 1$  and  $incy \neq M\_Y$ , it is  $Y_{iy:iy+n-1, jy:jy}$ .
- $\alpha$  and  $\beta$  are scalars.

and:

- For PDSYMV, submatrix  $A$  is real symmetric.
- For PZHEMV, submatrix  $A$  is complex Hermitian.

In the following two cases, no computation is performed and the subroutine returns after doing some parameter checking:

- $n = 0$
- $\alpha$  is zero and  $\beta$  is one.

See references [15] and [16].

Table 50. Data Types

$\alpha, \beta, A, x, y$	Subprogram
Long-precision real	PDSYMV
Long-precision complex	PZHEMV

### Syntax

<b>Fortran</b>	CALL PDSYMV   PZHEMV ( <i>uplo, n, alpha, a, ia, ja, desc_a, x, ix, jx, desc_x, incx, beta, y, iy, jy, desc_y, incy</i> )
<b>C and C++</b>	pdsymv   pzhemv ( <i>uplo, n, alpha, a, ia, ja, desc_a, x, ix, jx, desc_x, incx, beta, y, iy, jy, desc_y, incy</i> );

### On Entry

*uplo* indicates whether the upper or lower triangular part of the global submatrix  $A$  is referenced, where:

If *uplo* = 'U', the upper triangular part is referenced.

If *uplo* = 'L', the lower triangular part is referenced.

Scope: **global**

Specified as: a single character; *uplo* = 'U' or 'L'.

*n* is the number of rows and columns in submatrix  $A$  and the number of elements in vectors  $x$  and  $y$  used in the computation.

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

*alpha* is the scalar  $\alpha$ .

Scope: **global**

Specified as: a number of the data type indicated in Table 50 on page 154.

*a* is the local part of the global real symmetric or complex Hermitian matrix *A*. This identifies the **first element** of the local array *A*. This subroutine computes the location of the first element of the local subarray used, based on *ia*, *ja*, *desc\_a*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading LOCp(*ia*+*n*-1) by LOCq(*ja*+*n*-1) part of the local array *A* must contain the local pieces of the leading *ia*+*n*-1 by *ja*+*n*-1 part of the global matrix, and:

- If *uplo* = 'U', the leading  $n \times n$  upper triangular part of the global submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  must contain the upper triangular part of the submatrix, and the strictly lower triangular part is not referenced.
- If *uplo* = 'L', the leading  $n \times n$  lower triangular part of the global submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  must contain the lower triangular part of the submatrix, and the strictly upper triangular part is not referenced.

Scope: **local**

Specified as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 50 on page 154. Details about the square block-cyclic data distribution of global matrix *A* are stored in *desc\_a*.

*ia* is the row index of the global matrix *A*, identifying the first row of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq M\_A$  and  $ia+n-1 \leq M\_A$ .

*ja* is the column index of the global matrix *A*, identifying the first column of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$  and  $ja+n-1 \leq N\_A$ .

*desc\_a* is the array descriptor for global matrix *A*, described in the following table:

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global

<i>desc_a</i>	Name	Description	Limits	Scope
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq \text{RSRC\_A} < p$	Global
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq \text{CSRC\_A} < q$	Global
9	LLD_A	The leading dimension of the local array	$\text{LLD\_A} \geq \max(1, \text{LOCp}(\text{M\_A}))$	Local

Specified as: an array of (at least) length 9, containing fullword integers.

$x$  is the local part of the global matrix  $X$ . This identifies the **first element** of the local array  $X$ . This subroutine computes the location of the first element of the local subarray used, based on  $ix$ ,  $jx$ ,  $desc\_x$ ,  $p$ ,  $q$ ,  $myrow$ , and  $mycol$ ; therefore:

- If  $incx = \text{M\_X}$ , the leading  $\text{LOCp}(ix)$  by  $\text{LOCq}(jx+n-1)$  part of the local array  $X$  must contain the local pieces of the leading  $ix$  by  $jx+n-1$  part of the global matrix.
- If  $incx = 1$  and  $incx \neq \text{M\_X}$ , the leading  $\text{LOCp}(ix+n-1)$  by  $\text{LOCq}(jx)$  part of the local array  $X$  must contain the local pieces of the leading  $ix+n-1$  by  $jx$  part of the global matrix.

Scope: **local**

Specified as: an  $\text{LLD\_X}$  by (at least)  $\text{LOCq}(\text{N\_X})$  array, containing numbers of the data type indicated in Table 50 on page 154. Details about the block-cyclic data distribution of the global matrix  $X$  are stored in  $desc\_x$ .

$ix$  has the following meaning:

If  $incx = \text{M\_X}$ , it indicates which row of global matrix  $X$  is used for vector  $x$ .

If  $incx = 1$  and  $incx \neq \text{M\_X}$ , it is the row index of global matrix  $X$ , identifying the first element of vector  $x$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ix \leq \text{M\_X}$  and:

If  $incx = 1$  and  $incx \neq \text{M\_X}$ , then  $ix+n-1 \leq \text{M\_X}$ .

$jx$  has the following meaning:

If  $incx = \text{M\_X}$ , it is the column index of global matrix  $X$ , identifying the first element of vector  $x$ .

If  $incx = 1$  and  $incx \neq \text{M\_X}$ , it indicates which column of global matrix  $X$  is used for vector  $x$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq jx \leq \text{N\_X}$  and:

If  $incx = \text{M\_X}$ , then  $jx+n-1 \leq \text{N\_X}$ .

$desc\_x$  is the array descriptor for global matrix  $X$ , described in the following table:

<i>desc_x</i>	Name	Description	Limits	Scope
1	DTYPE_X	Descriptor type	$\text{DTYPE\_X}=1$	Global

<i>desc_x</i>	Name	Description	Limits	Scope
2	CTXT_X	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_X	Number of rows in the global matrix	If $n = 0$ : $M\_X \geq 0$ Otherwise: $M\_X \geq 1$	Global
4	N_X	Number of columns in the global matrix	If $n = 0$ : $N\_X \geq 0$ Otherwise: $N\_X \geq 1$	Global
5	MB_X	Row block size	$MB\_X \geq 1$	Global
6	NB_X	Column block size	$NB\_X \geq 1$	Global
7	RSRC_X	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_X < p$	Global
8	CSRC_X	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_X < q$	Global
9	LLD_X	The leading dimension of the local array	$LLD\_X \geq \max(1, LOCp(M\_X))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

*incx* is the stride for global vector  $x$ .

Scope: **global**

Specified as: a fullword integer;  $incx = 1$  or  $incx = M\_X$ , where:

If  $incx = M\_X$ , then  $x$  is a row-distributed vector.

If  $incx = 1$  and  $incx \neq M\_X$ , then  $x$  is a column-distributed vector.

*beta* is the scalar  $\beta$ .

Scope: **global**

Specified as: a number of the data type indicated in Table 50 on page 154.

*y* is the local part of the global matrix  $Y$ . This identifies the **first element** of the local array  $Y$ . This subroutine computes the location of the first element of the local subarray used, based on *iy*, *jy*, *desc\_y*, *p*, *q*, *myrow*, and *mycol*; therefore:

- If  $incy = M\_Y$ , the leading  $LOCp(iy)$  by  $LOCq(jy+n-1)$  part of the local array  $Y$  must contain the local pieces of the leading  $iy$  by  $jy+n-1$  part of the global matrix.
- If  $incy = 1$  and  $incy \neq M\_Y$ , the leading  $LOCp(iy+n-1)$  by  $LOCq(jy)$  part of the local array  $Y$  must contain the local pieces of the leading  $iy+n-1$  by  $jy$  part of the global matrix.

When  $\beta$  is zero, *y* need not be set on input.

Scope: **local**

Specified as: an LLD\_Y by (at least) LOCq(N\_Y) array, containing numbers of the data type indicated in Table 50 on page 154. Details about the block-cyclic data distribution of the global matrix *Y* are stored in *desc\_y*.

*iy* has the following meaning:

If *incy* = M\_Y, it indicates which row of global matrix *Y* is used for vector *y*.

If *incy* = 1 and *incy* ≠ M\_Y, it is the row index of global matrix *Y*, identifying the first element of vector *y*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq iy \leq M\_Y$  and:

If *incy* = 1 and *incy* ≠ M\_Y, then  $iy+n-1 \leq M\_Y$ .

*jy* has the following meaning:

If *incy* = M\_Y, it is the column index of global matrix *Y*, identifying the first element of vector *y*.

If *incy* = 1 and *incy* ≠ M\_Y, it indicates which column of global matrix *Y* is used for vector *y*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq jy \leq N\_Y$  and:

If *incy* = M\_Y, then  $jy+n-1 \leq N\_Y$ .

*desc\_y* is the array descriptor for global matrix *Y*, described in the following table:

<i>desc_y</i>	Name	Description	Limits	Scope
1	DTYPE_Y	Descriptor type	DTYPE_Y=1	Global
2	CTXT_Y	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_Y	Number of rows in the global matrix	If $n = 0$ : $M\_Y \geq 0$ Otherwise: $M\_Y \geq 1$	Global
4	N_Y	Number of columns in the global matrix	If $n = 0$ : $N\_Y \geq 0$ Otherwise: $N\_Y \geq 1$	Global
5	MB_Y	Row block size	$MB\_Y \geq 1$	Global
6	NB_Y	Column block size	$NB\_Y \geq 1$	Global
7	RSRC_Y	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_Y < p$	Global
8	CSRC_Y	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_Y < q$	Global
9	LLD_Y	The leading dimension of the local array	$LLD\_Y \geq \max(1, LOCp(M\_Y))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.



*incy* is the stride for global vector *y*.

Scope: **global**

Specified as: a fullword integer; *incy* = 1 or *incy* = *M\_X*, where:

If *incy* = *M\_Y*, then *y* is a row-distributed vector.

If *incy* = 1 and *incy*  $\neq$  *M\_Y*, then *y* is a column-distributed vector.

### On Return

*y* is the updated local part of the global matrix *Y*, containing the results of the computation.

Scope: **local**

Returned as: an LLD\_Y by (at least) LOCq(N\_Y) array, containing numbers of the data type indicated in Table 50 on page 154.

## Notes and Coding Rules

1. These subroutines accept lowercase letters for the *uplo* argument.
2. The matrix and vectors must have no common elements; otherwise, results are unpredictable.
3. The imaginary parts of the diagonal elements of a complex Hermitian matrix *A* are assumed to be zero, so you do not have to set these values.
4. The NUMROC utility subroutine can be used to determine the values of LOCp(M\_) and LOCq(N\_) used in the argument descriptions above. For details, see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28 and “NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process” on page 868.
5. For suggested block sizes, see “Coding Tips for Optimizing Parallel Performance” on page 77.
6. The following values must be equal: CTXT\_A = CTXT\_X = CTXT\_Y.
7. The global matrix *A* must be distributed using a square block-cyclic distribution; that is, MB\_A = NB\_A.
8. The block row and block column offsets of the global matrix *A* must be equal; that is, mod(*ia*−1, MB\_A) = mod(*ja*−1, NB\_A).
9. The vectors *x* and *y* must be distributed along the same axis—that is, they must both be row distributed or column distributed, where:
  - *incx* = *M\_X* and *incy* = *M\_Y* for row distribution
  - *incx* = 1(  $\neq$  *M\_X*) and *incy* = 1(  $\neq$  *M\_Y*) for column distribution
10. If *incx* = *M\_X* and *incy* = *M\_Y*, then (in the process grid) the process column containing the first column of the submatrix *A* must also contain the first column of the submatrices *X* and *Y*; that is:
  - *iacol* = *ixcol*
  - *iacol* = *iycol*
  - where:
    - *iacol* = mod((((*ja*−1)/NB\_A)+CSRC\_A), *q*)
    - *ixcol* = mod((((*jx*−1)/NB\_X)+CSRC\_X), *q*)
    - *iycol* = mod((((*jy*−1)/NB\_Y)+CSRC\_Y), *q*)
11. If *incx* = 1(  $\neq$  *M\_X*) and *incy* = 1(  $\neq$  *M\_Y*), then (in the process grid) the process row containing the first row of the submatrix *A* must also contain the first row of the submatrices *X* and *Y*; that is:
  - *iarow* = *ixrow*

- $iarow = iyrow$
- where:
  - $iarow = \text{mod}(\text{mod}(((ia-1)/MB\_A)+RSRC\_A), p)$
  - $ixrow = \text{mod}(\text{mod}(((ix-1)/MB\_X)+RSRC\_X), p)$
  - $iyrow = \text{mod}(\text{mod}(((iy-1)/MB\_Y)+RSRC\_Y), p)$
- 12. If  $incx = M\_X$ :
  - The block column offset of  $x$  must be equal to the block column offset of  $A$ ; that is,  $\text{mod}(jx-1, NB\_X) = \text{mod}(ja-1, NB\_A)$ .
  - The following block sizes must be equal:  $NB\_X = NB\_A$ .
- 13. If  $incx = 1$  ( $\neq M\_X$ ):
  - The block row offset of  $x$  must be equal to the block column offset of  $A$ ; that is,  $\text{mod}(ix-1, MB\_X) = \text{mod}(ja-1, NB\_A)$ .
  - The following block sizes must be equal:  $MB\_X = NB\_A$ .
- 14. If  $incy = M\_Y$ :
  - The block column offset of  $y$  must be equal to the block row offset of  $A$ ; that is,  $\text{mod}(jy-1, NB\_Y) = \text{mod}(ia-1, MB\_A)$ .
  - The following block sizes must be equal:  $NB\_Y = MB\_A$ .
- 15. If  $incy = 1$  ( $\neq M\_Y$ ):
  - The block row offset of  $y$  must be equal to the block row offset of  $A$ ; that is,  $\text{mod}(iy-1, MB\_Y) = \text{mod}(ia-1, MB\_A)$ .
  - The following block sizes must be equal:  $MB\_Y = MB\_A$ .

## Error Conditions

### Computational Errors

None

### Resource Errors

Unable to allocate work space

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1.  $DTYPE\_A$  is invalid.
2.  $DTYPE\_X$  is invalid.
3.  $DTYPE\_Y$  is invalid.

#### Stage 2:

1.  $CTXT\_A$  is invalid.

#### Stage 3:

1. This subroutine was called from outside the process grid.

#### Stage 4:

1.  $uplo \neq 'U'$  or  $'L'$
2.  $n < 0$
3.  $MB\_X < 1$
4.  $NB\_X < 1$
5.  $M\_X < 0$  and  $n = 0$ ;  $M\_X < 1$  otherwise
6.  $N\_X < 0$  and  $n = 0$ ;  $N\_X < 1$  otherwise
7.  $RSRC\_X < 0$  or  $RSRC\_X \geq p$
8.  $CSRC\_X < 0$  or  $CSRC\_X \geq q$
9.  $CTXT\_A \neq CTXT\_X$

10.  $ix < 1$
11.  $jx < 1$
12.  $MB\_Y < 1$
13.  $NB\_Y < 1$
14.  $M\_Y < 0$  and  $n = 0$ ;  $M\_Y < 1$  otherwise
15.  $N\_Y < 0$  and  $n = 0$ ;  $N\_Y < 1$  otherwise
16.  $RSRC\_Y < 0$  or  $RSRC\_Y \geq p$
17.  $CSRC\_Y < 0$  or  $CSRC\_Y \geq q$
18.  $CTXT\_A \neq CTXT\_Y$
19.  $iy < 1$
20.  $jy < 1$
21.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
22.  $CSRC\_A < 0$  or  $CSRC\_A \geq q$
23.  $M\_A < 0$  and ( $m = 0$  and  $n = 0$ );  $M\_A < 1$  otherwise
24.  $N\_A < 0$  and ( $m = 0$  and  $n = 0$ );  $N\_A < 1$  otherwise
25.  $NB\_A < 1$
26.  $MB\_A < 1$
27.  $ja < 1$
28.  $ia < 1$

**Stage 5:**

1.  $MB\_A \neq NB\_A$   
If  $n \neq 0$ :
2.  $ix > M\_X$
3.  $jx > N\_X$
4.  $iy > M\_Y$
5.  $jy > N\_Y$
6.  $ia+n-1 > M\_A$
7.  $ja+n-1 > N\_A$   
If  $incx = M\_X$  and  $incy = M\_Y$ :
8.  $NB\_A \neq NB\_X$
9.  $MB\_A \neq NB\_Y$
10.  $\text{mod}(jx-1, NB\_X) \neq \text{mod}(ja-1, NB\_A)$
11.  $\text{mod}(jy-1, NB\_Y) \neq \text{mod}(ia-1, MB\_A)$
12.  $n \neq 0$  and  $jx+n-1 > N\_X$
13.  $n \neq 0$  and  $jy+n-1 > N\_Y$   
If  $incx = 1(\neq M\_X)$  and  $incy = 1(\neq M\_Y)$ :
14.  $NB\_A \neq MB\_X$
15.  $MB\_A \neq MB\_Y$
16.  $\text{mod}(ix-1, MB\_X) \neq \text{mod}(ja-1, NB\_A)$
17.  $\text{mod}(iy-1, MB\_Y) \neq \text{mod}(ia-1, MB\_A)$
18.  $n \neq 0$  and  $ix+n-1 > M\_X$
19.  $n \neq 0$  and  $iy+n-1 > M\_Y$   
Otherwise:
20.  $incx \neq M\_X$  and  $incx \neq 1$
21.  $incy \neq M\_Y$  and  $incy \neq 1$

**Stage 6:**

1.  $LLD\_A < \max(1, \text{LOCp}(M\_A))$
2.  $LLD\_X < \max(1, \text{LOCp}(M\_X))$
3.  $LLD\_Y < \max(1, \text{LOCp}(M\_Y))$
4.  $\text{mod}(ia-1, MB\_A) \neq \text{mod}(ja-1, NB\_A)$
5. If  $incx = M\_X$  and  $incy = M\_Y$ , then (in the process grid) the process column containing the first column of the submatrix  $A$  does not contain the first column of the submatrices  $X$  and  $Y$ ; that is:

- $ixcol \neq iacol$
- $iycol \neq iacol$
- where:
  - $iacol = \text{mod}(((ja-1)/NB\_A)+CSRC\_A), q)$
  - $ixcol = \text{mod}(((jx-1)/NB\_X)+CSRC\_X), q)$
  - $iycol = \text{mod}(((jy-1)/NB\_Y)+CSRC\_Y), q)$
- 6. If  $incx = 1 (\neq M\_X)$  and  $incy = 1 (\neq M\_Y)$ , then (in the process grid) the process row containing the first row of the submatrix  $A$  does not contain the first row of the submatrices  $X$  and  $Y$ ; that is:
  - $ixrow \neq iarow$
  - $iyrow \neq iarow$
  - where:
    - $iarow = \text{mod}(((ia-1)/MB\_A)+RSRC\_A), p)$
    - $ixrow = \text{mod}(((ix-1)/MB\_X)+RSRC\_X), p)$
    - $iyrow = \text{mod}(((iy-1)/MB\_Y)+RSRC\_Y), p)$

## Examples

### Example 1

This example computes  $y = \alpha Ax + \beta y$  using a  $2 \times 2$  process grid.

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONXTX)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      UPLO   N   ALPHA   A   IA   JA   DESC_A   X   IX   JX
      |     |     |     |   |   |   |         |   |   |
CALL PDSYMV( 'U' , 8 , 1.0D0 , A , 1 , 1 , DESC_A , X , 1 , 1 ,

      DESC_X   INCX   BETA   Y   IY   JY   DESC_Y   INCY
      |         |     |     |   |   |   |         |
      DESC_X , 1 , 0.0D0 , Y , 1 , 1 , DESC_Y , 1 )
```

	Desc_A	Desc_X	Desc_Y
DTYPE_	1	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	8	8	8
N_	8	1	1
MB_	2	2	2
NB_	2	1	1
RSRC_	0	0	0
CSRC_	0	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>	See below <sup>2</sup>

#### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:
 

```
LLD_A = MAX(1,NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_X = MAX(1,NUMROC(M_X, MB_X, MYROW, RSRC_X, NPROW))
LLD_Y = MAX(1,NUMROC(M_Y, MB_Y, MYROW, RSRC_Y, NPROW))
```

In this example,  $LLD\_A = LLD\_X = LLD\_Y = 4$  on all processes.

Global real symmetric matrix  $A$  of order 8 with block size  $2 \times 2$ :

B,D	0	1	2	3
0	$\begin{bmatrix} 0.0 & -1.0 \\ . & 1.0 \end{bmatrix}$	$\begin{bmatrix} -1.0 & 0.0 \\ 0.0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 0.0 \\ 0.0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 0.0 \\ 0.0 & 1.0 \end{bmatrix}$
1	$\begin{bmatrix} . & . \\ . & . \end{bmatrix}$	$\begin{bmatrix} -1.0 & -1.0 \\ . & -1.0 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 0.0 \\ 1.0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{bmatrix}$
2	$\begin{bmatrix} . & . \\ . & . \end{bmatrix}$	$\begin{bmatrix} . & . \\ . & . \end{bmatrix}$	$\begin{bmatrix} -1.0 & 0.0 \\ . & 1.0 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 0.0 \\ 0.0 & 0.0 \end{bmatrix}$
3	$\begin{bmatrix} . & . \\ . & . \end{bmatrix}$	$\begin{bmatrix} . & . \\ . & . \end{bmatrix}$	$\begin{bmatrix} . & . \\ . & . \end{bmatrix}$	$\begin{bmatrix} 0.0 & 0.0 \\ . & 0.0 \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$
3		

Local arrays for  $A$ :

p,q	0	1
0	$\begin{bmatrix} 0.0 & -1.0 & 0.0 & 0.0 \\ . & 1.0 & 0.0 & 1.0 \\ . & . & -1.0 & 0.0 \\ . & . & . & 1.0 \end{bmatrix}$	$\begin{bmatrix} -1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 1.0 \\ . & . & 0.0 & 0.0 \\ . & . & 0.0 & 0.0 \end{bmatrix}$
1	$\begin{bmatrix} . & . & 0.0 & 0.0 \\ . & . & 1.0 & 1.0 \\ . & . & . & . \\ . & . & . & . \end{bmatrix}$	$\begin{bmatrix} -1.0 & -1.0 & 1.0 & 0.0 \\ . & -1.0 & 0.0 & 1.0 \\ . & . & 0.0 & 0.0 \\ . & . & . & 0.0 \end{bmatrix}$

Global vector  $x$  of size  $8 \times 1$  with block size 2:

B,D	0
0	$\begin{bmatrix} 1.0 \\ 1.0 \end{bmatrix}$
1	$\begin{bmatrix} 1.0 \\ 1.0 \end{bmatrix}$
2	$\begin{bmatrix} 1.0 \\ 1.0 \end{bmatrix}$
3	$\begin{bmatrix} 1.0 \\ 1.0 \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0	--
0	$P_{00}$	$P_{01}$

2		
1	$P_{10}$	$P_{11}$
3		

Local arrays for  $x$ :

p,q	0
	1.0
	1.0
0	1.0
	1.0
	1.0
	1.0
1	1.0
	1.0

**Output:**

Global vector  $y$  of size  $8 \times 1$  with block size 2:

B,D	0
0	-2.0
	3.0
1	-2.0
	2.0
2	0.0
	3.0
3	1.0
	2.0

The following is the  $2 \times 2$  process grid:

B,D	0	--
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$
3		

Local arrays for  $y$ :

p,q	0
	-2.0
	3.0
0	0.0
	3.0
	-2.0
	2.0
1	1.0
	2.0

## Example 2

This example computes  $y = \alpha Ax + \beta y$  using a  $2 \times 2$  process grid.

**Note:** The imaginary parts of the diagonal elements of a complex Hermitian matrix are assumed to be zero, so you do not have to set these values.

**Call Statements and Input:**

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      UPLO   N   ALPHA   A   IA   JA   DESC_A   X   IX   JX
      |     |   |     |   |   |   |     |   |   |
CALL PZHEMV( 'U' , 6 , ALPHA , A , 1 , 1 , DESC_A , X , 1 , 1 ,

      DESC_X   INCX   BETA   Y   IY   JY   DESC_Y   INCY
      |       |     |     |   |   |   |       |
      DESC_X , 1 , BETA , Y , 1 , 1 , DESC_Y , 1 )

ALPHA = (1.0,0.0)

BETA  = (0.0,0.0)
```

	Desc_A	Desc_X	Desc_Y
DTYPE_	1	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	6	6	6
N_	6	1	1
MB_	2	2	2
NB_	2	1	1
RSRC_	0	0	0
CSRC_	0	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>	See below <sup>2</sup>

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:

```
LLD_A = MAX(1,NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_X = MAX(1,NUMROC(M_X, MB_X, MYROW, RSRC_X, NPROW))
LLD_Y = MAX(1,NUMROC(M_Y, MB_Y, MYROW, RSRC_Y, NPROW))
```

In this example, LLD\_A = LLD\_X = LLD\_Y = 4 on P<sub>00</sub> and P<sub>01</sub> and  
LLD\_A = LLD\_X = LLD\_Y = 2 on P<sub>10</sub> and P<sub>11</sub>.

Global complex Hermitian matrix *A* of order 6 with block size 2 × 2:

B,D	0	1	2
0	$\begin{pmatrix} (0.0, 0.0) & (-1.0, 1.0) \\ & (2.0, 0.0) \end{pmatrix}$	$\begin{pmatrix} (-1.0, -2.0) & (0.0, 3.0) \\ (5.0, 4.0) & (2.0, 0.0) \end{pmatrix}$	$\begin{pmatrix} (2.0, 1.0) & (1.0, 0.0) \\ (-1.0, -1.0) & (0.0, 2.0) \end{pmatrix}$
1	$\begin{pmatrix} . & . \\ . & . \end{pmatrix}$	$\begin{pmatrix} (0.0, 0.0) & (-1.0, 0.0) \\ & (4.0, 0.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 0.0) & (1.0, 1.0) \\ (2.0, -1.0) & (-1.0, -1.0) \end{pmatrix}$
2	$\begin{pmatrix} . & . \\ . & . \end{pmatrix}$	$\begin{pmatrix} . & . \\ . & . \end{pmatrix}$	$\begin{pmatrix} (-1.0, 0.0) & (0.0, 2.0) \\ & (1.0, 0.0) \end{pmatrix}$

The following is the  $2 \times 2$  process grid for  $A$ :

B,D	0 2	1
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

Local arrays for  $A$ :

p,q	0	1
0	$\begin{pmatrix} 0.0, & . \\ . & 2.0, & . \end{pmatrix} \begin{pmatrix} -1.0, & 1.0 \\ 2.0, & . \end{pmatrix} \begin{pmatrix} 2.0, & 1.0 \\ -1.0, & -1.0 \end{pmatrix} \begin{pmatrix} 1.0, & 0.0 \\ 0.0, & 2.0 \end{pmatrix}$	$\begin{pmatrix} -1.0, & -2.0 \\ 5.0, & 4.0 \end{pmatrix} \begin{pmatrix} 0.0, & 3.0 \\ 2.0, & 0.0 \end{pmatrix}$
1	$\begin{pmatrix} 1.0, & 2.0 \\ -2.0, & -3.0 \end{pmatrix} \begin{pmatrix} 0.0, & 1.0 \\ 1.0, & 0.0 \end{pmatrix}$	$\begin{pmatrix} 0.0, & . \\ . & 4.0, & . \end{pmatrix} \begin{pmatrix} -1.0, & . \\ 4.0, & . \end{pmatrix}$

Global vector  $x$  of size  $6 \times 1$  with block size 2:

B,D	0
0	$\begin{pmatrix} -1.0, & 1.0 \\ 2.0, & 1.0 \end{pmatrix}$
1	$\begin{pmatrix} 1.0, & 2.0 \\ -2.0, & -3.0 \end{pmatrix}$
2	$\begin{pmatrix} 0.0, & 1.0 \\ 1.0, & 0.0 \end{pmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0	--
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

Local arrays for  $x$ :

p,q	0
0	$\begin{pmatrix} -1.0, & 1.0 \\ 2.0, & 1.0 \\ 0.0, & 1.0 \\ 1.0, & 0.0 \end{pmatrix}$
1	$\begin{pmatrix} 1.0, & 2.0 \\ -2.0, & -3.0 \end{pmatrix}$

**Output:**

Global vector  $y$  of size  $6 \times 1$  with block size 2:

B,D	0
0	$\begin{pmatrix} 9.0, & -7.0 \\ 0.0, & 11.0 \end{pmatrix}$
1	$\begin{pmatrix} 16.0, & -2.0 \\ -2.0, & -8.0 \end{pmatrix}$



2

-----

( -5.0, -3.0)

( 12.0, -1.0)

The following is the  $2 \times 2$  process grid:

B,D	0	--
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for *y*:

p,q	0
	-----
	( 9.0, -7.0)
	( 0.0, 11.0)
0	( -5.0, -3.0)
	( 12.0, -1.0)
	-----
	( 12.0, -2.0)
1	( -2.0, -8.0)

## PDGER, PZGERC, and PZGERU — Rank-One Update of a General Matrix

### Purpose

PDGER and PZGERU compute the following rank-one update:

- $A \leftarrow \alpha x y^T + A$

PZGERC computes the following rank-one update:

- $A \leftarrow \alpha x y^H + A$

where, in the formula above:

- $A$  represents the global general submatrix  $A_{ia:ia+m-1, ja:ja+n-1}$ .
- $x$  represents the global vector:
  - For  $incx = M\_X$ , it is  $X_{ix:ix, jx:jx+m-1}$ .
  - For  $incx = 1$  and  $incx \neq M\_X$ , it is  $X_{ix:ix+m-1, jx:jx}$ .
- $y$  represents the global vector:
  - For  $incy = M\_Y$ , it is  $Y_{iy:iy, jy:jy+n-1}$ .
  - For  $incy = 1$  and  $incy \neq M\_Y$ , it is  $Y_{iy:iy+n-1, jy:jy}$ .
- $\alpha$  is a scalar.

**Note:** No data should be moved to form  $y^T$  or  $y^H$ ; that is, the vector  $y$  should always be stored in its untransposed form.

In the following three cases, no computation is performed and the subroutine returns after doing some parameter checking:

- $m = 0$
- $n = 0$
- $\alpha$  is zero.

See references [15] and [16].

Table 51. Data Types

$\alpha, A, x, y$	Subprogram
Long-precision real	PDGER
Long-precision complex	PZGERC and PZGERU

### Syntax

#### On Entry

$m$  is the number of rows in submatrix  $A$  and the number of elements in vector  $x$  used in the computation.

Scope: **global**

Specified as: a fullword integer;  $m \geq 0$ .

$n$  is the number of columns in submatrix  $A$  and the number of elements in vector  $y$  used in the computation.

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

$alpha$  is the scalar  $\alpha$ .

Scope: **global**

Specified as: a number of the data type indicated in Table 51 on page 168.

$x$  is the local part of the global matrix  $X$ . This identifies the **first element** of the local array  $X$ . This subroutine computes the location of the first element of the local subarray used, based on  $ix$ ,  $jx$ ,  $desc\_x$ ,  $p$ ,  $q$ ,  $myrow$ , and  $mycol$ ; therefore:

- If  $incx = M\_X$ , the leading  $LOCp(ix)$  by  $LOCq(jx+m-1)$  part of the local array  $X$  must contain the local pieces of the leading  $ix$  by  $jx+m-1$  part of the global matrix.
- If  $incx = 1$  and  $incx \neq M\_X$ , the leading  $LOCp(ix+m-1)$  by  $LOCq(jx)$  part of the local array  $X$  must contain the local pieces of the leading  $ix+m-1$  by  $jx$  part of the global matrix.

Scope: **local**

Specified as: an  $LLD\_X$  by (at least)  $LOCq(N\_X)$  array, containing numbers of the data type indicated in Table 51 on page 168. Details about the block-cyclic data distribution of the global matrix  $X$  are stored in  $desc\_x$ .

$ix$  has the following meaning:

If  $incx = M\_X$ , it indicates which row of global matrix  $X$  is used for vector  $x$ .

If  $incx = 1$  and  $incx \neq M\_X$ , it is the row index of global matrix  $X$ , identifying the first element of vector  $x$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ix \leq M\_X$  and:

If  $incx = 1$  and  $incx \neq M\_X$ , then  $ix+m-1 \leq M\_X$ .

$jx$  has the following meaning:

If  $incx = M\_X$ , it is the column index of global matrix  $X$ , identifying the first element of vector  $x$ .

If  $incx = 1$  and  $incx \neq M\_X$ , it indicates which column of global matrix  $X$  is used for vector  $x$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq jx \leq N\_X$  and:

If  $incx = M\_X$ , then  $jx+m-1 \leq N\_X$ .

$desc\_x$  is the array descriptor for global matrix  $X$ , described in the following table:

$desc\_x$	Name	Description	Limits	Scope
1	DTYPE_X	Descriptor type	DTYPE_X=1	Global
2	CTXT_X	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_X	Number of rows in the global matrix	If $m = 0$ : $M\_X \geq 0$ Otherwise: $M\_X \geq 1$	Global
4	N_X	Number of columns in the global matrix	If $m = 0$ : $N\_X \geq 0$ Otherwise: $N\_X \geq 1$	Global

## PDGER, PZGERC, and PZGERU

<i>desc_x</i>	Name	Description	Limits	Scope
5	MB_X	Row block size	$MB\_X \geq 1$	Global
6	NB_X	Column block size	$NB\_X \geq 1$	Global
7	RSRC_X	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_X < p$	Global
8	CSRC_X	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_X < q$	Global
9	LLD_X	The leading dimension of the local array	$LLD\_X \geq \max(1, LOCp(M\_X))$	Local

Specified as: an array of (at least) length 9, containing fullword integers.

*incx* is the stride for global vector  $x$ .

Scope: **global**

Specified as: a fullword integer;  $incx = 1$  or  $incx = M\_X$ , where:

If  $incx = M\_X$ , then  $x$  is a row-distributed vector.

If  $incx = 1$  and  $incx \neq M\_X$ , then  $x$  is a column-distributed vector.

*y* is the local part of the global matrix  $Y$ . This identifies the **first element** of the local array  $Y$ . This subroutine computes the location of the first element of the local subarray used, based on *iy*, *jy*, *desc\_y*, *p*, *q*, *myrow*, and *mycol*; therefore:

- If  $incy = M\_Y$ , the leading  $LOCp(iy)$  by  $LOCq(jy+n-1)$  part of the local array  $Y$  must contain the local pieces of the leading  $iy$  by  $jy+n-1$  part of the global matrix.
- If  $incy = 1$  and  $incy \neq M\_Y$ , the leading  $LOCp(iy+n-1)$  by  $LOCq(jy)$  part of the local array  $Y$  must contain the local pieces of the leading  $iy+n-1$  by  $jy$  part of the global matrix.

**Note:** No data should be moved to form  $y^T$  or  $y^H$ ; that is, the vector  $y$  should always be stored in its untransposed form.

Scope: **local**

Specified as: an  $LLD\_Y$  by (at least)  $LOCq(N\_Y)$  array, containing numbers of the data type indicated in Table 51 on page 168. Details about the block-cyclic data distribution of the global matrix  $Y$  are stored in *desc\_y*.

*iy* has the following meaning:

If  $incy = M\_Y$ , it indicates which row of global matrix  $Y$  is used for vector  $y$ .

If  $incy = 1$  and  $incy \neq M\_Y$ , it is the row index of global matrix  $Y$ , identifying the first element of vector  $y$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq iy \leq M\_Y$  and:

If  $incy = 1$  and  $incy \neq M\_Y$ , then  $iy+n-1 \leq M\_Y$ .

*jy* has the following meaning:

If  $incy = M\_Y$ , it is the column index of global matrix  $Y$ , identifying the first element of vector  $y$ .

If  $incy = 1$  and  $incy \neq M\_Y$ , it indicates which column of global matrix  $Y$  is used for vector  $y$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq jy \leq N\_Y$  and:

If  $incy = M\_Y$ , then  $jy+n-1 \leq N\_Y$ .

$desc\_y$  is the array descriptor for global matrix  $Y$ , described in the following table:

$desc\_y$	Name	Description	Limits	Scope
1	DTYPE_Y	Descriptor type	DTYPE_Y=1	Global
2	CTXT_Y	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_Y	Number of rows in the global matrix	If $n = 0$ : $M\_Y \geq 0$ Otherwise: $M\_Y \geq 1$	Global
4	N_Y	Number of columns in the global matrix	If $n = 0$ : $N\_Y \geq 0$ Otherwise: $N\_Y \geq 1$	Global
5	MB_Y	Row block size	$MB\_Y \geq 1$	Global
6	NB_Y	Column block size	$NB\_Y \geq 1$	Global
7	RSRC_Y	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_Y < p$	Global
8	CSRC_Y	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_Y < q$	Global
9	LLD_Y	The leading dimension of the local array	$LLD\_Y \geq \max(1, LOCp(M\_Y))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

$incy$  is the stride for global vector  $y$ .

Scope: **global**

Specified as: a fullword integer;  $incy = 1$  or  $incy = M\_X$ , where:

If  $incy = M\_Y$ , then  $y$  is a row-distributed vector.

If  $incy = 1$  and  $incy \neq M\_Y$ , then  $y$  is a column-distributed vector.

$a$  is the local part of the global general matrix  $A$ . This identifies the **first element** of the local array  $A$ . This subroutine computes the location of the first element of the local subarray used, based on  $ia$ ,  $ja$ ,  $desc\_a$ ,  $p$ ,  $q$ ,  $myrow$ , and  $mycol$ ; therefore, the leading  $LOCp(ia+m-1)$  by  $LOCq(ja+n-1)$  part of the local array  $A$  must contain the local pieces of the leading  $ia+m-1$  by  $ja+n-1$  part of the global matrix.

Scope: **local**

## PDGER, PZGERC, and PZGERU

Specified as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 51 on page 168. Details about the block-cyclic data distribution of global matrix *A* are stored in *desc\_a*.

*ia* is the row index of the global matrix *A*, identifying the first row of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq M\_A$  and  $ia+m-1 \leq M\_A$ .

*ja* is the column index of the global matrix *A*, identifying the first column of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$  and  $ja+n-1 \leq N\_A$ .

*desc\_a* is the array descriptor for global matrix *A*, described in the following table:

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $m = 0$ or $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	If $m = 0$ or $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_A < p$	Global
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < q$	Global
9	LLD_A	The leading dimension of the local array	$LLD\_A \geq \max(1, LOCp(M\_A))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

### On Return

*a* is the updated local part of the global matrix *A*, containing the results of the computation.

Scope: **local**

Returned as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 51 on page 168.

## Notes and Coding Rules

1. The matrix and vectors must have no common elements; otherwise, results are unpredictable.
2. The NUMROC utility subroutine can be used to determine the values of  $LOCp(M\_)$  and  $LOCq(N\_)$  used in the argument descriptions above. For details, see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28 and “NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process” on page 868.
3. For suggested block sizes, see “Coding Tips for Optimizing Parallel Performance” on page 77.
4. The following values must be equal:  $CTXT\_A = CTXT\_X = CTXT\_Y$ .
5. If  $incx = M\_X$ :
  - The block column offset of  $x$  must be equal to the block row offset of  $A$ ; that is,  $mod(jx-1, NB\_X) = mod(ia-1, MB\_A)$ .
  - The following block sizes must be equal:  $NB\_X = MB\_A$ .
6. If  $incx = 1 ( \neq M\_X )$ :
  - In the process grid, the process row containing the first row of the submatrix  $A$  must also contain the first row of the submatrix  $X$ ; that is,  $iarrow = ixrow$ , where:
    - $iarrow = mod((((ia-1)/MB\_A)+RSRC\_A), p)$
    - $ixrow = mod((((ix-1)/MB\_X)+RSRC\_X), p)$
  - The block row offset of  $x$  must be equal to the block row offset of  $A$ ; that is,  $mod(ix-1, MB\_X) = mod(ia-1, MB\_A)$ .
  - The following block sizes must be equal:  $MB\_X = MB\_A$ .
7. If  $incy = M\_Y$ :
  - In the process grid, the process column containing the first column of the submatrix  $A$  must also contain the first column of the submatrix  $Y$ ; that is,  $iacol = iycol$ , where:
    - $iacol = mod((((ja-1)/NB\_A)+CSRC\_A), q)$
    - $iycol = mod((((jy-1)/NB\_Y)+CSRC\_Y), q)$
  - The block column offset of  $y$  must be equal to the block column offset of  $A$ ; that is,  $mod(jy-1, NB\_Y) = mod(ja-1, NB\_A)$ .
  - The following block sizes must be equal:  $NB\_Y = NB\_A$ .
8. If  $incy = 1 ( \neq M\_Y )$ :
  - The block row offset of  $y$  must be equal to the block column offset of  $A$ ; that is,  $mod(iy-1, MB\_Y) = mod(ja-1, NB\_A)$ .
  - The following block sizes must be equal:  $MB\_Y = NB\_A$ .

## Error Conditions

### Computational Errors

None

### Resource Errors

Unable to allocate work space

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1.  $DTYPE\_A$  is invalid.
2.  $DTYPE\_X$  is invalid.

3. DTYPE\_Y is invalid.

## Stage 2:

1. CTEXT\_A is invalid.

## Stage 3:

1. This subroutine was called from outside the process grid.

## Stage 4:

1.  $m < 0$
2.  $n < 0$
3.  $M\_X < 0$  and  $m = 0$ ;  $M\_X < 1$  otherwise
4.  $N\_X < 0$  and  $m = 0$ ;  $N\_X < 1$  otherwise
5.  $MB\_X < 1$
6.  $NB\_X < 1$
7.  $RSRC\_X < 0$  or  $RSRC\_X \geq p$
8.  $CSRC\_X < 0$  or  $CSRC\_X \geq q$
9.  $CTXT\_A \neq CTXT\_X$
10.  $ix < 1$
11.  $jx < 1$
12.  $M\_Y < 0$  and  $n = 0$ ;  $M\_Y < 1$  otherwise
13.  $N\_Y < 0$  and  $n = 0$ ;  $N\_Y < 1$  otherwise
14.  $MB\_Y < 1$
15.  $NB\_Y < 1$
16.  $RSRC\_Y < 0$  or  $RSRC\_Y \geq p$
17.  $CSRC\_Y < 0$  or  $CSRC\_Y \geq q$
18.  $CTXT\_A \neq CTXT\_Y$
19.  $iy < 1$
20.  $jy < 1$
21.  $M\_A < 0$  and  $(m = 0 \text{ or } n = 0)$ ;  $M\_A < 1$  otherwise
22.  $N\_A < 0$  and  $(m = 0 \text{ or } n = 0)$ ;  $N\_A < 1$  otherwise
23.  $MB\_A < 1$
24.  $NB\_A < 1$
25.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
26.  $CSRC\_A < 0$  or  $CSRC\_A \geq q$
27.  $ia < 1$
28.  $ja < 1$

## Stage 5: If $m \neq 0$ and $n \neq 0$ :

1.  $ia > M\_A$
2.  $ja > N\_A$
3.  $ia+m-1 > M\_A$
4.  $ja+n-1 > N\_A$
- If  $m \neq 0$ :
5.  $ix > M\_X$
6.  $jx > N\_X$
- If  $n \neq 0$ :
7.  $iy > M\_Y$
8.  $jy > N\_Y$
- If  $incx = M\_X$ :
9.  $NB\_X \neq MB\_A$
10.  $\text{mod}(jx-1, NB\_X) \neq \text{mod}(ia-1, MB\_A)$
11.  $m \neq 0$  and  $jx+m-1 > N\_X$
- If  $incx = 1 (\neq M\_X)$ :
12.  $MB\_X \neq MB\_A$



13.  $\text{mod}(ix-1, MB\_X) \neq \text{mod}(ia-1, MB\_A)$
14.  $m \neq 0$  and  $ix+m-1 > M\_X$   
Otherwise:
15.  $incx \neq M\_X$  and  $incx \neq 1$   
If  $incy = M\_Y$ :
16.  $NB\_Y \neq NB\_A$
17.  $\text{mod}(jy-1, NB\_Y) \neq \text{mod}(ja-1, NB\_A)$
18.  $n \neq 0$  and  $jy+n-1 > N\_Y$   
If  $incy = 1$  ( $\neq M\_Y$ ):
19.  $MB\_Y \neq NB\_A$
20.  $\text{mod}(iy-1, MB\_Y) \neq \text{mod}(ja-1, NB\_A)$
21.  $n \neq 0$  and  $iy+n-1 > M\_Y$   
Otherwise:
22.  $incy \neq M\_Y$  and  $incy \neq 1$

**Stage 6:**

1. If  $incx = 1$  ( $\neq M\_X$ ), then (in the process grid) the process row containing the first row of the submatrix  $A$  does not contain the first row of the submatrix  $X$ ; that is,  $iarow \neq ixrow$ , where:
  - $iarow = \text{mod}(((ia-1)/MB\_A)+RSRC\_A), p)$
  - $ixrow = \text{mod}(((ix-1)/MB\_X)+RSRC\_X), p)$
2. If  $incy = M\_Y$ , then (in the process grid) the process column containing the first column of the submatrix  $A$  does not contain the first column of the submatrix  $Y$ ; that is,  $iacol \neq iycol$ , where:
  - $iacol = \text{mod}(((ja-1)/NB\_A)+CSRC\_A), q)$
  - $iycol = \text{mod}(((jy-1)/NB\_Y)+CSRC\_Y), q)$
3.  $LLD\_A < \max(1, LOCp(M\_A))$
4.  $LLD\_X < \max(1, LOCp(M\_X))$
5.  $LLD\_Y < \max(1, LOCp(M\_Y))$

**Examples****Example 1**

This example computes  $A = \alpha xy^T + A$  using a  $2 \times 2$  process grid. It uses a global submatrix  $A$  within a global matrix  $A$  by specifying  $ia = 2$  and  $ja = 2$ . It uses vector  $x$ , which is a column-distributed vector within a column of global matrix  $X$ , by specifying  $incx = 1$ ,  $ix = 2$ , and  $jx = 1$ . It uses vector  $y$ , which is a row-distributed vector within a row of global matrix  $Y$ , by specifying  $incy = M\_Y = 1$ ,  $iy = 1$ , and  $jy = 2$ .

**Call Statements and Input:**

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      M   N   ALPHA   X   IX   JX   DESC_X   INCX   Y   IY   JY
CALL PDGER( 9 , 9 , 1.0D0 , X , 2 , 1 , DESC_X , 1 , Y , 1 , 2 ,
            DESC_Y   INCY   A   IA   JA   DESC_A
            DESC_Y , 1 , A , 2 , 2 , DESC_A )
```

	Desc_A	Desc_X	Desc_Y
DTYPE_	1	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	10	11	1
N_	10	1	11
MB_	4	4	1
NB_	4	1	4
RSRC_	0	0	0
CSRC_	0	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>	See below <sup>2</sup>

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.

2. Each process should set the LLD\_ as follows:

```
LLD_A = MAX(1,NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_X = MAX(1,NUMROC(M_X, MB_X, MYROW, RSRC_X, NPROW))
LLD_Y = MAX(1,NUMROC(M_Y, MB_Y, MYROW, RSRC_Y, NPROW))
```

In this example, LLD\_A = 6 on P<sub>00</sub> and P<sub>01</sub>, LLD\_A = 4 on P<sub>10</sub> and P<sub>11</sub>,  
LLD\_X = 7 on P<sub>00</sub>, LLD\_X = 4 on P<sub>10</sub>, LLD\_Y = 1 on P<sub>00</sub> and P<sub>01</sub>.

After the global matrix *A* is distributed over the process grid, only a portion of the global data structure is used—that is, global submatrix *A*. Following is the global 9 × 9 submatrix *A*, starting at row 2 and column 2 in global general 10 × 10 matrix *A* with block size 4 × 4:

B,D	0	1	2
0	$\begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & 12.0 & 22.0 & 32.0 \\ \cdot & 13.0 & 23.0 & 33.0 \\ \cdot & 14.0 & 24.0 & 34.0 \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ 42.0 & 52.0 & 62.0 & 72.0 \\ 43.0 & 53.0 & 63.0 & 73.0 \\ 44.0 & 54.0 & 64.0 & 74.0 \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot \\ 82.0 & 92.0 \\ 83.0 & 93.0 \\ 84.0 & 94.0 \end{bmatrix}$
1	$\begin{bmatrix} \cdot & 15.0 & 25.0 & 35.0 \\ \cdot & 16.0 & 26.0 & 36.0 \\ \cdot & 17.0 & 27.0 & 37.0 \\ \cdot & 18.0 & 28.0 & 38.0 \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ 45.0 & 55.0 & 65.0 & 75.0 \\ 46.0 & 56.0 & 66.0 & 76.0 \\ 47.0 & 57.0 & 67.0 & 77.0 \\ 48.0 & 58.0 & 68.0 & 78.0 \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot \\ 85.0 & 95.0 \\ 86.0 & 96.0 \\ 87.0 & 97.0 \\ 88.0 & 98.0 \end{bmatrix}$
2	$\begin{bmatrix} \cdot & 19.0 & 29.0 & 39.0 \\ \cdot & 20.0 & 30.0 & 40.0 \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ 49.0 & 59.0 & 69.0 & 79.0 \\ 50.0 & 60.0 & 70.0 & 80.0 \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot \\ 89.0 & 99.0 \\ 90.0 & 100.0 \end{bmatrix}$

The following is the 2 × 2 process grid:

B,D	0 2	1
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for *A*:

p,q	0	1
0	$\begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & 12.0 & 22.0 & 32.0 & 82.0 & 92.0 \\ \cdot & 13.0 & 23.0 & 33.0 & 83.0 & 93.0 \\ \cdot & 14.0 & 24.0 & 34.0 & 84.0 & 94.0 \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ 42.0 & 52.0 & 62.0 & 72.0 \\ 43.0 & 53.0 & 63.0 & 73.0 \\ 44.0 & 54.0 & 64.0 & 74.0 \end{bmatrix}$

		.	19.0	29.0	39.0	89.0	99.0		49.0	59.0	69.0	79.0
		.	20.0	30.0	40.0	90.0	100.0		50.0	60.0	70.0	80.0
-----												
		.	15.0	25.0	35.0	85.0	95.0		45.0	55.0	65.0	75.0
		.	16.0	26.0	36.0	86.0	96.0		46.0	56.0	66.0	76.0
1		.	17.0	27.0	37.0	87.0	97.0		47.0	57.0	67.0	77.0
		.	18.0	28.0	38.0	88.0	98.0		48.0	58.0	68.0	78.0

After the global matrix  $X$  is distributed over the process grid, only a portion of the global data structure is used—that is, global vector  $x$ , which is a column-distributed vector. Following is the global vector  $x$  of size  $9 \times 1$ , starting at row 2 and column 1 in  $11 \times 1$  global matrix  $X$  with block size  $4 \times 1$ :

B,D	0
	[
	.
	1.0
0	1.0
	1.0
	-----
	1.0
	1.0
1	1.0
	1.0
	-----
	1.0
2	1.0
	.
	]

The following is the  $2 \times 2$  process grid:

B,D	0	--
	-----	-----
0	$P_{00}$	$P_{01}$
2		
	-----	-----
1	$P_{10}$	$P_{11}$

Local arrays for  $x$ :

p,q	0
	-----
	.
	1.0
	1.0
0	1.0
	1.0
	1.0
	.
	-----
	1.0
	1.0
1	1.0
	1.0

After the global matrix  $Y$  is distributed over the process grid, only a portion of the global data structure is used—that is, global vector  $y$ , which is a row-distributed vector. Following is the global vector  $y$  of size  $1 \times 9$ , starting at row 1 and column 2 in  $1 \times 11$  global matrix  $Y$  with block size  $1 \times 4$ :

B,D	0	1	2
0	[	.	2.0 3.0 4.0   5.0 6.0 7.0 8.0   9.0 10.0 . ]

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
--	$P_{10}$	$P_{11}$

Local arrays for  $y$ :

p,q	0	1
0	. 2.0 3.0 4.0 9.0 10.0 .	5.0 6.0 7.0 8.0

### Output:

After the global matrix  $A$  is distributed over the process grid, only a portion of the global data structure is used—that is, global submatrix  $A$ . Following is the global  $9 \times 9$  submatrix  $A$ , starting at row 2 and column 2 in global general  $10 \times 10$  matrix  $A$  with block size  $4 \times 4$ :

B,D	0	1	2
0	<div> <div>. 14.0 25.0 36.0</div> <div>47.0 58.0 69.0 80.0</div> <div>91.0 102.0</div> </div>	<div> <div>47.0 58.0 69.0 80.0</div> <div>48.0 59.0 70.0 81.0</div> <div>49.0 60.0 71.0 82.0</div> </div>	<div> <div>91.0 102.0</div> <div>92.0 103.0</div> <div>93.0 104.0</div> </div>
1	<div> <div>. 17.0 28.0 39.0</div> <div>50.0 61.0 72.0 83.0</div> <div>94.0 105.0</div> </div>	<div> <div>50.0 61.0 72.0 83.0</div> <div>51.0 62.0 73.0 84.0</div> <div>52.0 63.0 74.0 85.0</div> </div>	<div> <div>94.0 105.0</div> <div>95.0 106.0</div> <div>96.0 107.0</div> </div>
2	<div> <div>. 21.0 32.0 43.0</div> <div>54.0 65.0 76.0 87.0</div> <div>98.0 109.0</div> </div>	<div> <div>54.0 65.0 76.0 87.0</div> <div>55.0 66.0 77.0 88.0</div> <div>99.0 110.0</div> </div>	<div> <div>98.0 109.0</div> <div>99.0 110.0</div> </div>

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

Local arrays for  $A$ :

p,q	0	1
0	. 14.0 25.0 36.0 91.0 102.0 . 15.0 26.0 37.0 92.0 103.0 . 16.0 27.0 38.0 93.0 104.0 . 21.0 32.0 43.0 98.0 109.0 . 22.0 33.0 44.0 99.0 110.0	47.0 58.0 69.0 80.0 48.0 59.0 70.0 81.0 49.0 60.0 71.0 82.0 54.0 65.0 76.0 87.0 55.0 66.0 77.0 88.0
1	. 17.0 28.0 39.0 94.0 105.0 . 18.0 29.0 40.0 95.0 106.0 . 19.0 30.0 41.0 96.0 107.0 . 20.0 31.0 42.0 97.0 108.0	50.0 61.0 72.0 83.0 51.0 62.0 73.0 84.0 52.0 63.0 74.0 85.0 53.0 64.0 75.0 86.0

### Example 2

This example computes  $A = \alpha xy^H + A$  using a  $2 \times 2$  process grid. It uses a global submatrix  $A$  within a global matrix  $A$  by specifying  $ia = 2$  and  $ja = 2$ . It uses vector  $x$ , which is a column-distributed vector within a column of global matrix  $X$ ,

by specifying  $incx = 1$ ,  $ix = 2$ , and  $jx = 1$ . It uses vector  $y$ , which is a row-distributed vector within a row of global matrix  $Y$ , by specifying  $incy = M\_Y = 1$ ,  $iy = 1$ , and  $jy = 2$ .

### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      M   N   ALPHA   X   IX   JX   DESC_X   INCX   Y   IY   JY
      |   |   |       |   |   |   |       |   |   |   |
CALL PZGERC( 9 , 9 , ALPHA , X , 2 , 1 , DESC_X , 1 , Y , 1 , 2 ,

      DESC_Y   INCY   A   IA   JA   DESC_A
      |       |       |   |   |   |
      DESC_Y , 1 , A , 2 , 2 , DESC_A )

ALPHA = (1.0, -1.0)
```

	Desc_A	Desc_X	Desc_Y
DTYPE_	1	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	10	11	1
N_	10	1	11
MB_	4	4	1
NB_	4	1	4
RSRC_	0	0	0
CSRC_	0	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>	See below <sup>2</sup>

### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.

2. Each process should set the LLD\_ as follows:

```
LLD_A = MAX(1, NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_X = MAX(1, NUMROC(M_X, MB_X, MYROW, RSRC_X, NPROW))
LLD_Y = MAX(1, NUMROC(M_Y, MB_Y, MYROW, RSRC_Y, NPROW))
```

In this example, LLD\_A = 6 on P<sub>00</sub> and P<sub>01</sub>, LLD\_A = 4 on P<sub>10</sub> and P<sub>11</sub>,  
LLD\_X = 7 on P<sub>00</sub>, LLD\_X = 4 on P<sub>10</sub>, LLD\_Y = 1 on P<sub>00</sub> and P<sub>01</sub>.

After the global matrix  $A$  is distributed over the process grid, only a portion of the global data structure is used—that is, global submatrix  $A$ . Following is the global  $9 \times 9$  submatrix  $A$ , starting at row 2 and column 2 in global general  $10 \times 10$  matrix  $A$  with block size  $4 \times 4$ :

## PDGER, PZGERC, and PZGERU

B,D	0	1	2
0	$\begin{bmatrix} \cdot & (12.0, 2.0) & (22.0, 1.0) & (32.0, 0.0) \\ \cdot & (13.0, 2.0) & (23.0, 1.0) & (33.0, 0.0) \\ \cdot & (14.0, 2.0) & (24.0, 1.0) & (34.0, 0.0) \end{bmatrix}$	$\begin{bmatrix} (42.0, -1.0) & (52.0, -2.0) & (62.0, 2.0) & (72.0, 1.0) \\ (43.0, -1.0) & (53.0, -2.0) & (63.0, 2.0) & (73.0, 1.0) \\ (44.0, -1.0) & (54.0, -2.0) & (64.0, 2.0) & (74.0, 1.0) \end{bmatrix}$	$\begin{bmatrix} (82.0, 0.0) & (92.0, -1.0) \\ (83.0, 0.0) & (93.0, -1.0) \\ (84.0, 0.0) & (94.0, -1.0) \end{bmatrix}$
1	$\begin{bmatrix} \cdot & (15.0, 2.0) & (25.0, 1.0) & (35.0, 0.0) \\ \cdot & (16.0, 2.0) & (26.0, 1.0) & (36.0, 0.0) \\ \cdot & (17.0, 2.0) & (27.0, 1.0) & (37.0, 0.0) \\ \cdot & (18.0, 2.0) & (28.0, 1.0) & (38.0, 0.0) \end{bmatrix}$	$\begin{bmatrix} (45.0, -1.0) & (55.0, -2.0) & (65.0, 2.0) & (75.0, 1.0) \\ (46.0, -1.0) & (56.0, -2.0) & (66.0, 2.0) & (76.0, 1.0) \\ (47.0, -1.0) & (57.0, -2.0) & (67.0, 2.0) & (77.0, 1.0) \\ (48.0, -1.0) & (58.0, -2.0) & (68.0, 2.0) & (78.0, 1.0) \end{bmatrix}$	$\begin{bmatrix} (85.0, 0.0) & (95.0, -1.0) \\ (86.0, 0.0) & (96.0, -1.0) \\ (87.0, 0.0) & (97.0, -1.0) \\ (88.0, 0.0) & (98.0, -1.0) \end{bmatrix}$
2	$\begin{bmatrix} \cdot & (19.0, 2.0) & (29.0, 1.0) & (39.0, 0.0) \\ \cdot & (20.0, 2.0) & (30.0, 1.0) & (40.0, 0.0) \end{bmatrix}$	$\begin{bmatrix} (49.0, -1.0) & (59.0, -2.0) & (69.0, 2.0) & (79.0, 1.0) \\ (50.0, -1.0) & (60.0, -2.0) & (70.0, 2.0) & (80.0, 1.0) \end{bmatrix}$	$\begin{bmatrix} (89.0, 0.0) & (99.0, -1.0) \\ (90.0, 0.0) & (100.0, -1.0) \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for A:

p,q	0	1
0	$\begin{bmatrix} \cdot & (12.0, 2.0) & (22.0, 1.0) & (32.0, 0.0) & (82.0, 0.0) & (92.0, -1.0) \\ \cdot & (13.0, 2.0) & (23.0, 1.0) & (33.0, 0.0) & (83.0, 0.0) & (93.0, -1.0) \\ \cdot & (14.0, 2.0) & (24.0, 1.0) & (34.0, 0.0) & (84.0, 0.0) & (94.0, -1.0) \\ \cdot & (19.0, 2.0) & (29.0, 1.0) & (39.0, 0.0) & (89.0, 0.0) & (99.0, -1.0) \\ \cdot & (20.0, 2.0) & (30.0, 1.0) & (40.0, 0.0) & (90.0, 0.0) & (100.0, -1.0) \end{bmatrix}$	$\begin{bmatrix} (42.0, -1.0) & (52.0, -2.0) & (62.0, 2.0) & (72.0, 1.0) \\ (43.0, -1.0) & (53.0, -2.0) & (63.0, 2.0) & (73.0, 1.0) \\ (44.0, -1.0) & (54.0, -2.0) & (64.0, 2.0) & (74.0, 1.0) \\ (49.0, -1.0) & (59.0, -2.0) & (69.0, 2.0) & (79.0, 1.0) \\ (50.0, -1.0) & (60.0, -2.0) & (70.0, 2.0) & (80.0, 1.0) \end{bmatrix}$
1	$\begin{bmatrix} \cdot & (15.0, 2.0) & (25.0, 1.0) & (35.0, 0.0) & (85.0, 0.0) & (95.0, -1.0) \\ \cdot & (16.0, 2.0) & (26.0, 1.0) & (36.0, 0.0) & (86.0, 0.0) & (96.0, -1.0) \\ \cdot & (17.0, 2.0) & (27.0, 1.0) & (37.0, 0.0) & (87.0, 0.0) & (97.0, -1.0) \\ \cdot & (18.0, 2.0) & (28.0, 1.0) & (38.0, 0.0) & (88.0, 0.0) & (98.0, -1.0) \end{bmatrix}$	$\begin{bmatrix} (45.0, -1.0) & (55.0, -2.0) & (65.0, 2.0) & (75.0, 1.0) \\ (46.0, -1.0) & (56.0, -2.0) & (66.0, 2.0) & (76.0, 1.0) \\ (47.0, -1.0) & (57.0, -2.0) & (67.0, 2.0) & (77.0, 1.0) \\ (48.0, -1.0) & (58.0, -2.0) & (68.0, 2.0) & (78.0, 1.0) \end{bmatrix}$

After the global matrix  $X$  is distributed over the process grid, only a portion of the global data structure is used—that is, global vector  $x$ , which is a column-distributed vector. Following is the global vector  $x$  of size  $9 \times 1$ , starting at row 2 and column 1 in  $11 \times 1$  global matrix  $X$  with block size  $4 \times 1$ :

B,D	0
0	$\begin{bmatrix} \cdot \\ (1.0, 4.0) \\ (1.0, 3.0) \\ (1.0, 2.0) \\ \cdot \\ (1.0, 1.0) \\ (1.0, 0.0) \\ (1.0, -1.0) \\ (1.0, -2.0) \\ \cdot \\ (1.0, -3.0) \\ (1.0, -4.0) \\ \cdot \end{bmatrix}$
1	
2	

The following is the  $2 \times 2$  process grid:

B,D	0	--
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for  $x$ :

p,q	0
0	.
	(1.0, 4.0)
	(1.0, 3.0)
	(1.0, 2.0)
	(1.0,-3.0)
	(1.0,-4.0)
1	.
	(1.0, 1.0)
	(1.0, 0.0)
	(1.0,-1.0)
	(1.0,-2.0)

After the global matrix  $Y$  is distributed over the process grid, only a portion of the global data structure is used—that is, global vector  $y$ , which is a row-distributed vector. Following is the global vector  $y$  of size  $1 \times 9$ , starting at row 1 and column 2 in  $1 \times 11$  global matrix  $Y$  with block size  $1 \times 4$ :

B,D	0	1	2						
0	$\left[ \begin{array}{cccc cccc} . & (2.0, 1.0) & (3.0, 1.0) & (4.0, 1.0) & (5.0, 1.0) & (6.0, 1.0) & (7.0, 1.0) & (8.0, 1.0) & (9.0, 1.0) & (10.0, 1.0) & . \end{array} \right]$								

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
--	$P_{10}$	$P_{11}$

Local arrays for  $y$ :

p,q	0	1
0	. (2.0, 1.0) (3.0, 1.0) (4.0, 1.0) (9.0, 1.0) (10.0, 1.0) .	(5.0, 1.0) (6.0, 1.0) (7.0, 1.0) (8.0, 1.0)

**Output:**

After the global matrix  $A$  is distributed over the process grid, only a portion of the global data structure is used—that is, global submatrix  $A$ . Following is the global  $9 \times 9$  submatrix  $A$ , starting at row 2 and column 2 in global general  $10 \times 10$  matrix  $A$  with block size  $4 \times 4$ :

B,D	0				1				2			
Г												
0	.	(25.0, 3.0)	(40.0, 5.0)	(55.0, 7.0)	(70.0, 9.0)	(85.0, 11.0)	(100.0, 18.0)	(115.0, 20.0)	(130.0, 22.0)	(145.0, 24.0)		
	.	(23.0, 2.0)	(37.0, 3.0)	(51.0, 4.0)	(65.0, 5.0)	(79.0, 6.0)	( 93.0, 12.0)	(107.0, 13.0)	(121.0, 14.0)	(135.0, 15.0)		
	.	(21.0, 1.0)	(34.0, 1.0)	(47.0, 1.0)	(60.0, 1.0)	(73.0, 1.0)	( 86.0, 6.0)	( 99.0, 6.0)	(112.0, 6.0)	(125.0, 6.0)		
1	.	(19.0, 0.0)	(31.0, -1.0)	(43.0, -2.0)	(55.0, -3.0)	(67.0, -4.0)	( 79.0, 0.0)	( 91.0, -1.0)	(103.0, -2.0)	(115.0, -3.0)		
	.	(17.0, -1.0)	(28.0, -3.0)	(39.0, -5.0)	(50.0, -7.0)	(61.0, -9.0)	( 72.0, -6.0)	( 83.0, -8.0)	( 94.0, -10.0)	(105.0, -12.0)		
	.	(15.0, -2.0)	(25.0, -5.0)	(35.0, -8.0)	(45.0, -11.0)	(55.0, -14.0)	( 65.0, -12.0)	( 75.0, -15.0)	( 85.0, -18.0)	( 95.0, -21.0)		
	.	(13.0, -3.0)	(22.0, -7.0)	(31.0, -11.0)	(40.0, -15.0)	(49.0, -19.0)	( 58.0, -18.0)	( 67.0, -22.0)	( 76.0, -26.0)	( 85.0, -30.0)		
2	.	(11.0, -4.0)	(19.0, -9.0)	(27.0, -14.0)	(35.0, -19.0)	(43.0, -24.0)	( 51.0, -24.0)	( 59.0, -29.0)	( 67.0, -34.0)	( 75.0, -39.0)		
	.	( 9.0, -5.0)	(16.0, -11.0)	(23.0, -17.0)	(30.0, -23.0)	(37.0, -29.0)	( 44.0, -30.0)	( 51.0, -36.0)	( 58.0, -42.0)	( 65.0, -48.0)		

The following is the  $2 \times 2$  process grid:

## PDGER, PZGERC, and PZGERU

B,D	0 2	1
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for A:

p,q	0	1
0	. (25.0, 3.0) (40.0, 5.0) (55.0, 7.0) (130.0, 22.0) (145.0, 24.0) . (23.0, 2.0) (37.0, 3.0) (51.0, 4.0) (121.0, 14.0) (135.0, 15.0) . (21.0, 1.0) (34.0, 1.0) (47.0, 1.0) (112.0, 6.0) (125.0, 6.0) . (11.0, -4.0) (19.0, -9.0) (27.0, -14.0) ( 67.0, -34.0) ( 75.0, -39.0) . ( 9.0, -5.0) (16.0, -11.0) (23.0, -17.0) ( 58.0, -42.0) ( 65.0, -48.0)	(70.0, 9.0) (85.0, 11.0) (100.0, 18.0) (115.0, 20.0) (65.0, 5.0) (79.0, 6.0) ( 93.0, 12.0) (107.0, 13.0) (60.0, 1.0) (73.0, 1.0) ( 86.0, 6.0) ( 99.0, 6.0) (35.0, -19.0) (43.0, -24.0) ( 51.0, -24.0) ( 59.0, -29.0) (30.0, -23.0) (37.0, -29.0) ( 44.0, -30.0) ( 51.0, -36.0)
1	. (19.0, 0.0) (31.0, -1.0) (43.0, -2.0) (103.0, -2.0) (115.0, -3.0) . (17.0, -1.0) (28.0, -3.0) (39.0, -5.0) ( 94.0, -10.0) (105.0, -12.0) . (15.0, -2.0) (25.0, -5.0) (35.0, -8.0) ( 85.0, -18.0) ( 95.0, -21.0) . (13.0, -3.0) (22.0, -7.0) (31.0, -11.0) ( 76.0, -26.0) ( 85.0, -30.0)	(55.0, -3.0) (67.0, -4.0) ( 79.0, 0.0) ( 91.0, -1.0) (50.0, -7.0) (61.0, -9.0) ( 72.0, -6.0) ( 83.0, -8.0) (45.0, -11.0) (55.0, -14.0) ( 65.0, -12.0) ( 75.0, -15.0) (40.0, -15.0) (49.0, -19.0) ( 58.0, -18.0) ( 67.0, -22.0)

### Example 3

This example computes  $A = \alpha xy^T + A$  using a  $2 \times 2$  process grid. It uses a global submatrix  $A$  within a global matrix  $A$  by specifying  $ia = 2$  and  $ja = 2$ . It uses vector  $x$ , which is a column-distributed vector within a column of global matrix  $X$ , by specifying  $incx = 1$ ,  $ix = 2$ , and  $jx = 1$ . It uses vector  $y$ , which is a row-distributed vector within a row of global matrix  $Y$ , by specifying  $incy = M\_Y = 1$ ,  $iy = 1$ , and  $jy = 2$ .

### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      M   N   ALPHA   X   IX   JX   DESC_X   INCX   Y   IY   JY
CALL PZGERU( 9 , 9 , ALPHA , X , 2 , 1 , DESC_X , 1 , Y , 1 , 2 ,

      DESC_Y   INCY   A   IA   JA   DESC_A
      |         |   |   |   |         |
DESC_Y , 1 , A , 2 , 2 , DESC_A )

ALPHA = (1.0,-1.0)
```

	Desc_A	Desc_X	Desc_Y
DTYPE_	1	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	10	11	1
N_	10	1	11
MB_	4	4	1
NB_	4	1	4
RSRC_	0	0	0
CSRC_	0	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>	See below <sup>2</sup>



**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.

2. Each process should set the LLD\_ as follows:

LLD\_A = MAX(1, NUMROC(M\_A, MB\_A, MYROW, RSRC\_A, NPROW))

LLD\_X = MAX(1, NUMROC(M\_X, MB\_X, MYROW, RSRC\_X, NPROW))

LLD\_Y = MAX(1, NUMROC(M\_Y, MB\_Y, MYROW, RSRC\_Y, NPROW))

In this example, LLD\_A = 6 on P<sub>00</sub> and P<sub>01</sub>, LLD\_A = 4 on P<sub>10</sub> and P<sub>11</sub>,

LLD\_X = 7 on P<sub>00</sub>, LLD\_X = 4 on P<sub>10</sub>, LLD\_Y = 1 on P<sub>00</sub> and P<sub>01</sub>.

After the global matrix *A* is distributed over the process grid, only a portion of the global data structure is used—that is, global submatrix *A*. Following is the global 9 × 9 submatrix *A*, starting at row 2 and column 2 in global general 10 × 10 matrix *A* with block size 4 × 4:

B,D	0	1	2
0	$\begin{pmatrix} \cdot & (12.0, 2.0) & (22.0, 1.0) & (32.0, 0.0) \\ \cdot & (13.0, 2.0) & (23.0, 1.0) & (33.0, 0.0) \\ \cdot & (14.0, 2.0) & (24.0, 1.0) & (34.0, 0.0) \end{pmatrix}$	$\begin{pmatrix} (42.0, -1.0) & (52.0, -2.0) & (62.0, 2.0) & (72.0, 1.0) \\ (43.0, -1.0) & (53.0, -2.0) & (63.0, 2.0) & (73.0, 1.0) \\ (44.0, -1.0) & (54.0, -2.0) & (64.0, 2.0) & (74.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (82.0, 0.0) & (92.0, -1.0) \\ (83.0, 0.0) & (93.0, -1.0) \\ (84.0, 0.0) & (94.0, -1.0) \end{pmatrix}$
1	$\begin{pmatrix} \cdot & (15.0, 2.0) & (25.0, 1.0) & (35.0, 0.0) \\ \cdot & (16.0, 2.0) & (26.0, 1.0) & (36.0, 0.0) \\ \cdot & (17.0, 2.0) & (27.0, 1.0) & (37.0, 0.0) \\ \cdot & (18.0, 2.0) & (28.0, 1.0) & (38.0, 0.0) \end{pmatrix}$	$\begin{pmatrix} (45.0, -1.0) & (55.0, -2.0) & (65.0, 2.0) & (75.0, 1.0) \\ (46.0, -1.0) & (56.0, -2.0) & (66.0, 2.0) & (76.0, 1.0) \\ (47.0, -1.0) & (57.0, -2.0) & (67.0, 2.0) & (77.0, 1.0) \\ (48.0, -1.0) & (58.0, -2.0) & (68.0, 2.0) & (78.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (85.0, 0.0) & (95.0, -1.0) \\ (86.0, 0.0) & (96.0, -1.0) \\ (87.0, 0.0) & (97.0, -1.0) \\ (88.0, 0.0) & (98.0, -1.0) \end{pmatrix}$
2	$\begin{pmatrix} \cdot & (19.0, 2.0) & (29.0, 1.0) & (39.0, 0.0) \\ \cdot & (20.0, 2.0) & (30.0, 1.0) & (40.0, 0.0) \end{pmatrix}$	$\begin{pmatrix} (49.0, -1.0) & (59.0, -2.0) & (69.0, 2.0) & (79.0, 1.0) \\ (50.0, -1.0) & (60.0, -2.0) & (70.0, 2.0) & (80.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (89.0, 0.0) & (99.0, -1.0) \\ (90.0, 0.0) & (100.0, -1.0) \end{pmatrix}$

The following is the 2 × 2 process grid:

B,D	0 2	1
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for *A*:

p,q	0	1
0	$\begin{pmatrix} \cdot & (12.0, 2.0) & (22.0, 1.0) & (32.0, 0.0) & (82.0, 0.0) & (92.0, -1.0) \\ \cdot & (13.0, 2.0) & (23.0, 1.0) & (33.0, 0.0) & (83.0, 0.0) & (93.0, -1.0) \\ \cdot & (14.0, 2.0) & (24.0, 1.0) & (34.0, 0.0) & (84.0, 0.0) & (94.0, -1.0) \\ \cdot & (19.0, 2.0) & (29.0, 1.0) & (39.0, 0.0) & (89.0, 0.0) & (99.0, -1.0) \\ \cdot & (20.0, 2.0) & (30.0, 1.0) & (40.0, 0.0) & (90.0, 0.0) & (100.0, -1.0) \end{pmatrix}$	$\begin{pmatrix} (42.0, -1.0) & (52.0, -2.0) & (62.0, 2.0) & (72.0, 1.0) \\ (43.0, -1.0) & (53.0, -2.0) & (63.0, 2.0) & (73.0, 1.0) \\ (44.0, -1.0) & (54.0, -2.0) & (64.0, 2.0) & (74.0, 1.0) \\ (49.0, -1.0) & (59.0, -2.0) & (69.0, 2.0) & (79.0, 1.0) \\ (50.0, -1.0) & (60.0, -2.0) & (70.0, 2.0) & (80.0, 1.0) \end{pmatrix}$
1	$\begin{pmatrix} \cdot & (15.0, 2.0) & (25.0, 1.0) & (35.0, 0.0) & (85.0, 0.0) & (95.0, -1.0) \\ \cdot & (16.0, 2.0) & (26.0, 1.0) & (36.0, 0.0) & (86.0, 0.0) & (96.0, -1.0) \\ \cdot & (17.0, 2.0) & (27.0, 1.0) & (37.0, 0.0) & (87.0, 0.0) & (97.0, -1.0) \\ \cdot & (18.0, 2.0) & (28.0, 1.0) & (38.0, 0.0) & (88.0, 0.0) & (98.0, -1.0) \end{pmatrix}$	$\begin{pmatrix} (45.0, -1.0) & (55.0, -2.0) & (65.0, 2.0) & (75.0, 1.0) \\ (46.0, -1.0) & (56.0, -2.0) & (66.0, 2.0) & (76.0, 1.0) \\ (47.0, -1.0) & (57.0, -2.0) & (67.0, 2.0) & (77.0, 1.0) \\ (48.0, -1.0) & (58.0, -2.0) & (68.0, 2.0) & (78.0, 1.0) \end{pmatrix}$

After the global matrix *X* is distributed over the process grid, only a portion of the global data structure is used—that is, global vector *x*, which is a column-distributed vector. Following is the global vector *x* of size 9 × 1, starting at row 2 and column 1 in 11 × 1 global matrix *X* with block size 4 × 1:

B,D	0
0	$\begin{pmatrix} \cdot \\ (1.0, 4.0) \\ (1.0, 3.0) \\ (1.0, 2.0) \\ \text{-----} \\ (1.0, 1.0) \end{pmatrix}$

## PDGER, PZGERC, and PZGERU

1	(1.0, 0.0)
	(1.0,-1.0)
	(1.0,-2.0)
-----	
2	(1.0,-3.0)
	(1.0,-4.0)
	.

The following is the  $2 \times 2$  process grid:

B,D	0	--
-----		
0	P <sub>00</sub>	P <sub>01</sub>
2		
-----		
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for  $x$ :

p,q	0
-----	
0	.
	(1.0, 4.0)
	(1.0, 3.0)
	(1.0, 2.0)
	(1.0,-3.0)
	(1.0,-4.0)
-----	
1	.
	(1.0, 1.0)
	(1.0, 0.0)
	(1.0,-1.0)
-----	
1	(1.0,-2.0)

After the global matrix  $Y$  is distributed over the process grid, only a portion of the global data structure is used—that is, global vector  $y$ , which is a row-distributed vector. Following is the global vector  $y$  of size  $1 \times 9$ , starting at row 1 and column 2 in  $1 \times 11$  global matrix  $Y$  with block size  $1 \times 4$ :

B,D	0	1	2
0	[ . (2.0, 1.0) (3.0, 1.0) (4.0, 1.0)   (5.0, 1.0) (6.0, 1.0) (7.0, 1.0) (8.0, 1.0)   (9.0, 1.0) (10.0, 1.0) . ]		

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
-----		
0	P <sub>00</sub>	P <sub>01</sub>
--	P <sub>10</sub>	P <sub>11</sub>

Local arrays for  $y$ :

p,q	0	1
-----		-----
0	[ . (2.0, 1.0) (3.0, 1.0) (4.0, 1.0) (9.0, 1.0) (10.0, 1.0) . ]	(5.0, 1.0) (6.0, 1.0) (7.0, 1.0) (8.0, 1.0)

### Output:

After the global matrix  $A$  is distributed over the process grid, only a portion of the global data structure is used—that is, global submatrix  $A$ . Following is the global

$9 \times 9$  submatrix  $A$ , starting at row 2 and column 2 in global general  $10 \times 10$  matrix  $A$  with block size  $4 \times 4$ :

B,D	0	1	2
0	$\begin{pmatrix} \cdot & \cdot & \cdot \\ (19.0, 13.0) & (34.0, 15.0) & (49.0, 17.0) \\ (19.0, 10.0) & (33.0, 11.0) & (47.0, 12.0) \\ (19.0, 7.0) & (32.0, 7.0) & (45.0, 7.0) \end{pmatrix}$	$\begin{pmatrix} \cdot & \cdot & \cdot \\ (64.0, 19.0) & (79.0, 21.0) & (94.0, 28.0) & (109.0, 30.0) \\ (61.0, 13.0) & (75.0, 14.0) & (89.0, 20.0) & (103.0, 21.0) \\ (58.0, 7.0) & (71.0, 7.0) & (84.0, 12.0) & (97.0, 12.0) \end{pmatrix}$	$\begin{pmatrix} \cdot & \cdot \\ (124.0, 32.0) & (139.0, 34.0) \\ (117.0, 22.0) & (131.0, 23.0) \\ (110.0, 12.0) & (123.0, 12.0) \end{pmatrix}$
1	$\begin{pmatrix} \cdot & \cdot & \cdot \\ (19.0, 4.0) & (31.0, 3.0) & (43.0, 2.0) \\ (19.0, 1.0) & (30.0, -1.0) & (41.0, -3.0) \\ (19.0, -2.0) & (29.0, -5.0) & (39.0, -8.0) \\ (19.0, -5.0) & (28.0, -9.0) & (37.0, -13.0) \end{pmatrix}$	$\begin{pmatrix} \cdot & \cdot & \cdot \\ (55.0, 1.0) & (67.0, 0.0) & (79.0, 4.0) & (91.0, 3.0) \\ (52.0, -5.0) & (63.0, -7.0) & (74.0, -4.0) & (85.0, -6.0) \\ (49.0, -11.0) & (59.0, -14.0) & (69.0, -12.0) & (79.0, -15.0) \\ (46.0, -17.0) & (55.0, -21.0) & (64.0, -20.0) & (73.0, -24.0) \end{pmatrix}$	$\begin{pmatrix} \cdot & \cdot \\ (103.0, 2.0) & (115.0, 1.0) \\ (96.0, -8.0) & (107.0, -10.0) \\ (89.0, -18.0) & (99.0, -21.0) \\ (82.0, -28.0) & (91.0, -32.0) \end{pmatrix}$
2	$\begin{pmatrix} \cdot \\ (19.0, -8.0) & (27.0, -13.0) & (35.0, -18.0) \\ (19.0, -11.0) & (26.0, -17.0) & (33.0, -23.0) \end{pmatrix}$	$\begin{pmatrix} \cdot \\ (43.0, -23.0) & (51.0, -28.0) & (59.0, -28.0) & (67.0, -33.0) \\ (40.0, -29.0) & (47.0, -35.0) & (54.0, -36.0) & (61.0, -42.0) \end{pmatrix}$	$\begin{pmatrix} \cdot \\ (75.0, -38.0) & (83.0, -43.0) \\ (68.0, -48.0) & (75.0, -54.0) \end{pmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

Local arrays for  $A$ :

p,q	0	1
0	$\begin{pmatrix} \cdot & \cdot & \cdot \\ (19.0, 13.0) & (34.0, 15.0) & (49.0, 17.0) & (124.0, 32.0) & (139.0, 34.0) \\ (19.0, 10.0) & (33.0, 11.0) & (47.0, 12.0) & (117.0, 22.0) & (131.0, 23.0) \\ (19.0, 7.0) & (32.0, 7.0) & (45.0, 7.0) & (110.0, 12.0) & (123.0, 12.0) \\ (19.0, -8.0) & (27.0, -13.0) & (35.0, -18.0) & (75.0, -38.0) & (83.0, -43.0) \\ (19.0, -11.0) & (26.0, -17.0) & (33.0, -23.0) & (68.0, -48.0) & (75.0, -54.0) \end{pmatrix}$	$\begin{pmatrix} \cdot & \cdot \\ (64.0, 19.0) & (79.0, 21.0) & (94.0, 28.0) & (109.0, 30.0) \\ (61.0, 13.0) & (75.0, 14.0) & (89.0, 20.0) & (103.0, 21.0) \\ (58.0, 7.0) & (71.0, 7.0) & (84.0, 12.0) & (97.0, 12.0) \\ (43.0, -23.0) & (51.0, -28.0) & (59.0, -28.0) & (67.0, -33.0) \\ (40.0, -29.0) & (47.0, -35.0) & (54.0, -36.0) & (61.0, -42.0) \end{pmatrix}$
1	$\begin{pmatrix} \cdot & \cdot & \cdot \\ (19.0, 4.0) & (31.0, 3.0) & (43.0, 2.0) & (103.0, 2.0) & (115.0, 1.0) \\ (19.0, 1.0) & (30.0, -1.0) & (41.0, -3.0) & (96.0, -8.0) & (107.0, -10.0) \\ (19.0, -2.0) & (29.0, -5.0) & (39.0, -8.0) & (89.0, -18.0) & (99.0, -21.0) \\ (19.0, -5.0) & (28.0, -9.0) & (37.0, -13.0) & (82.0, -28.0) & (91.0, -32.0) \end{pmatrix}$	$\begin{pmatrix} \cdot & \cdot \\ (55.0, 1.0) & (67.0, 0.0) & (79.0, 4.0) & (91.0, 3.0) \\ (52.0, -5.0) & (63.0, -7.0) & (74.0, -4.0) & (85.0, -6.0) \\ (49.0, -11.0) & (59.0, -14.0) & (69.0, -12.0) & (79.0, -15.0) \\ (46.0, -17.0) & (55.0, -21.0) & (64.0, -20.0) & (73.0, -24.0) \end{pmatrix}$

## PDSYR and PZHER — Rank-One Update of a Real Symmetric or a Complex Hermitian Matrix

### Purpose

PDSYR computes the following rank-one update:

$$\bullet A \leftarrow \alpha x x^T + A$$

PZHER computes the following rank-one update:

$$\bullet A \leftarrow \alpha x x^H + A$$

where, in the formula above:

- $A$  represents the global submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$ .
- $x$  represents the global vector:
  - For  $incx = M\_X$ , it is  $X_{ix:ix, jx:jx+n-1}$ .
  - For  $incx = 1$  and  $incx \neq M\_X$ , it is  $X_{ix:ix+n-1, jx:jx}$ .
- $\alpha$  is a scalar.

and:

- For PDSYR, submatrix  $A$  is real symmetric.
- For PZHER, submatrix  $A$  is complex Hermitian.

**Note:** No data should be moved to form  $x^T$  or  $x^H$ ; that is, the vector  $x$  should always be stored in its untransposed form.

In the following two cases, no computation is performed and the subroutine returns after doing some parameter checking:

- $n = 0$
- $\alpha$  is zero.

See references [15] and [16].

Table 52. Data Types

$A, x$	$\alpha$	Subprogram
Long-precision real	Long-precision real	PDSYR
Long-precision complex	Long-precision real	PZHER

### Syntax

<b>Fortran</b>	CALL PDSYR   PZHER ( <i>uplo</i> , <i>n</i> , <i>alpha</i> , <i>x</i> , <i>ix</i> , <i>jx</i> , <i>desc_x</i> , <i>incx</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> )
<b>C and C++</b>	pdsyr   pzher ( <i>uplo</i> , <i>n</i> , <i>alpha</i> , <i>x</i> , <i>ix</i> , <i>jx</i> , <i>desc_x</i> , <i>incx</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> );

### On Entry

*uplo* indicates whether the upper or lower triangular part of the global submatrix  $A$  is referenced, where:

If *uplo* = 'U', the upper triangular part is referenced.

If *uplo* = 'L', the lower triangular part is referenced.

Scope: **global**

Specified as: a single character; *uplo* = 'U' or 'L'.

$n$  is the number of rows and columns in submatrix  $A$  and the number of elements in vector  $x$  used in the computation.

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

$\alpha$  is the scalar  $\alpha$ .

Scope: **global**

Specified as: a number of the data type indicated in Table 52 on page 186.

$x$  is the local part of the global matrix  $X$ . This identifies the **first element** of the local array  $X$ . This subroutine computes the location of the first element of the local subarray used, based on  $ix$ ,  $jx$ ,  $desc\_x$ ,  $p$ ,  $q$ ,  $myrow$ , and  $mycol$ ; therefore:

- If  $incx = M\_X$ , the leading  $LOCp(ix)$  by  $LOCq(jx+n-1)$  part of the local array  $X$  must contain the local pieces of the leading  $ix$  by  $jx+n-1$  part of the global matrix.
- If  $incx = 1$  and  $incx \neq M\_X$ , the leading  $LOCp(ix+n-1)$  by  $LOCq(jx)$  part of the local array  $X$  must contain the local pieces of the leading  $ix+n-1$  by  $jx$  part of the global matrix.

**Note:** No data should be moved to form  $x^T$  or  $x^H$ ; that is, the vector  $x$  should always be stored in its untransposed form.

Scope: **local**

Specified as: an  $LLD\_X$  by (at least)  $LOCq(N\_X)$  array, containing numbers of the data type indicated in Table 52 on page 186. Details about the block-cyclic data distribution of the global matrix  $X$  are stored in  $desc\_x$ .

$ix$  has the following meaning:

If  $incx = M\_X$ , it indicates which row of global matrix  $X$  is used for vector  $x$ .

If  $incx = 1$  and  $incx \neq M\_X$ , it is the row index of global matrix  $X$ , identifying the first element of vector  $x$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ix \leq M\_X$  and:

If  $incx = 1$  and  $incx \neq M\_X$ , then  $ix+n-1 \leq M\_X$ .

$jx$  has the following meaning:

If  $incx = M\_X$ , it is the column index of global matrix  $X$ , identifying the first element of vector  $x$ .

If  $incx = 1$  and  $incx \neq M\_X$ , it indicates which column of global matrix  $X$  is used for vector  $x$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq jx \leq N\_X$  and:

If  $incx = M\_X$ , then  $jx+n-1 \leq N\_X$ .

$desc\_x$  is the array descriptor for global matrix  $X$ , described in the following table:

$desc\_x$	Name	Description	Limits	Scope
1	DTYPE_X	Descriptor type	DTYPE_X=1	Global

<i>desc_x</i>	Name	Description	Limits	Scope
2	CTXT_X	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_X	Number of rows in the global matrix	If $n = 0$ : $M\_X \geq 0$ Otherwise: $M\_X \geq 1$	Global
4	N_X	Number of columns in the global matrix	If $n = 0$ : $N\_X \geq 0$ Otherwise: $N\_X \geq 1$	Global
5	MB_X	Row block size	$MB\_X \geq 1$	Global
6	NB_X	Column block size	$NB\_X \geq 1$	Global
7	RSRC_X	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_X < p$	Global
8	CSRC_X	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_X < q$	Global
9	LLD_X	The leading dimension of the local array	$LLD\_X \geq \max(1, LOCp(M\_X))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

*incx* is the stride for global vector  $x$ .

Scope: **global**

Specified as: a fullword integer;  $incx = 1$  or  $incx = M\_X$ , where:

If  $incx = M\_X$ , then  $x$  is a row-distributed vector.

If  $incx = 1$  and  $incx \neq M\_X$ , then  $x$  is a column-distributed vector.

*a* is the local part of the global real symmetric or complex Hermitian matrix  $A$ . This identifies the **first element** of the local array  $A$ . This subroutine computes the location of the first element of the local subarray used, based on *ia*, *ja*, *desc\_a*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading  $LOCp(ia+n-1)$  by  $LOCq(ja+n-1)$  part of the local array  $A$  must contain the local pieces of the leading  $ia+n-1$  by  $ja+n-1$  part of the global matrix, and:

- If *uplo* = 'U', the leading  $n \times n$  upper triangular part of the global submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  must contain the upper triangular part of the submatrix, and the strictly lower triangular part is not referenced.
- If *uplo* = 'L', the leading  $n \times n$  lower triangular part of the global submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  must contain the lower triangular part of the submatrix, and the strictly upper triangular part is not referenced.

Scope: **local**

Specified as: an LLD\_A by (at least)  $LOCq(N\_A)$  array, containing numbers of the data type indicated in Table 52 on page 186. Details about the square block-cyclic data distribution of global matrix  $A$  are stored in *desc\_a*.

*ia* is the row index of the global matrix  $A$ , identifying the first row of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq M\_A$  and  $ia+n-1 \leq M\_A$ .

*ja* is the column index of the global matrix *A*, identifying the first column of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$  and  $ja+n-1 \leq N\_A$ .

*desc\_a* is the array descriptor for global matrix *A*, described in the following table:

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_A < p$	Global
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < q$	Global
9	LLD_A	The leading dimension of the local array	$LLD\_A \geq \max(1, LOCp(M\_A))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

### On Return

*a* is the updated local part of the global matrix *A*, containing the results of the computation.

Scope: **local**

Returned as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 52 on page 186.

## Notes and Coding Rules

1. These subroutines accept lowercase letters for the *uplo* argument.
2. The matrix and vector must have no common elements; otherwise, results are unpredictable.

3. The imaginary parts of the diagonal elements of the complex Hermitian matrix are assumed to be zero, so you do not have to set these values. On output, they are set to zero except when  $N$  is zero or  $\alpha$  is zero, in which case no computation is performed.
4. The NUMROC utility subroutine can be used to determine the values of  $LOCp(M\_)$  and  $LOCq(N\_)$  used in the argument descriptions above. For details, see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28 and “NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process” on page 868.
5. For suggested block sizes, see “Coding Tips for Optimizing Parallel Performance” on page 77.
6. The following values must be equal:  $CTXT\_A = CTXT\_X$ .
7. The global matrix  $A$  must be distributed using a square block-cyclic distribution; that is,  $MB\_A = NB\_A$ .
8. The block row and block column offsets of the global matrix  $A$  must be equal; that is,  $\text{mod}(ia-1, MB\_A) = \text{mod}(ja-1, NB\_A)$ .
9. If  $incx = M\_X$ :
  - In the process grid, the process column containing the first column of the submatrix  $A$  must also contain the first column of the submatrix  $X$ ; that is,  $iacol = ixcol$ , where:
    - $iacol = \text{mod}(((ja-1)/NB\_A)+CSRC\_A), q)$
    - $ixcol = \text{mod}(((jx-1)/NB\_X)+CSRC\_X), q)$
  - The block column offset of  $x$  must be equal to the block row offset of  $A$ ; that is,  $\text{mod}(jx-1, NB\_X) = \text{mod}(ia-1, MB\_A)$ .
  - The following block sizes must be equal:  $NB\_X = NB\_A$ .
10. If  $incx = 1 (\neq M\_X)$ :
  - In the process grid, the process row containing the first row of the submatrix  $A$  must also contain the first row of the submatrix  $X$ ; that is,  $iarow = ixrow$ , where:
    - $iarow = \text{mod}(((ia-1)/MB\_A)+RSRC\_A), p)$
    - $ixrow = \text{mod}(((ix-1)/MB\_X)+RSRC\_X), p)$
  - The block row offset of  $x$  must be equal to the block row offset of  $A$ ; that is,  $\text{mod}(ix-1, MB\_X) = \text{mod}(ia-1, MB\_A)$ .
  - The following block sizes must be equal:  $MB\_X = MB\_A$ .

## Error Conditions

### Computational Errors

None

### Resource Errors

Unable to allocate work space

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1.  $DTYPE\_A$  is invalid.
2.  $DTYPE\_X$  is invalid.

#### Stage 2:

1.  $CTXT\_A$  is invalid.



**Stage 3:**

1. This subroutine was called from outside the process grid.

**Stage 4:**

1.  $uplo \neq 'U' \text{ or } 'L'$
2.  $n < 0$
3.  $M\_X < 0$  and  $n = 0$ ;  $M\_X < 1$  otherwise
4.  $N\_X < 0$  and  $n = 0$ ;  $N\_X < 1$  otherwise
5.  $MB\_X < 1$
6.  $NB\_X < 1$
7.  $RSRC\_X < 0$  or  $RSRC\_X \geq p$
8.  $CSRC\_X < 0$  or  $CSRC\_X \geq q$
9.  $CTXT\_A \neq CTXT\_X$
10.  $ix < 1$
11.  $jx < 1$
12.  $M\_A < 0$  and  $n = 0$ ;  $M\_A < 1$  otherwise
13.  $N\_A < 0$  and  $n = 0$ ;  $N\_A < 1$  otherwise
14.  $MB\_A < 1$
15.  $NB\_A < 1$
16.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
17.  $CSRC\_A < 0$  or  $CSRC\_A \geq q$
18.  $ia < 1$
19.  $ja < 1$

**Stage 5:**

1.  $NB\_A \neq MB\_A$   
If  $n \neq 0$ :
2.  $ia > M\_A$
3.  $ja > N\_A$
4.  $ia+n-1 > M\_A$
5.  $ja+n-1 > N\_A$
6.  $ix > M\_X$
7.  $jx > N\_X$   
If  $incx = M\_X$ :
8.  $NB\_X \neq NB\_A$
9.  $\text{mod}(jx-1, NB\_X) \neq \text{mod}(ia-1, MB\_A)$
10.  $n \neq 0$  and  $jx+n-1 > N\_X$   
If  $incx = 1 (\neq M\_X)$ :
11.  $MB\_X \neq MB\_A$
12.  $\text{mod}(ix-1, MB\_X) \neq \text{mod}(ia-1, MB\_A)$
13.  $n \neq 0$  and  $ix+n-1 > M\_X$   
Otherwise:
14.  $incx \neq M\_X$  and  $incx \neq 1$

**Stage 6:**

1.  $\text{mod}(ja-1, NB\_A) \neq \text{mod}(ia-1, MB\_A)$
2. If  $incx = M\_X$ , then (in the process grid) the process column containing the first column of the submatrix  $A$  does not contain the first column of the submatrix  $X$ ; that is,  $iacol \neq ixcol$ , where:
  - $iacol = \text{mod}(((ja-1)/NB\_A)+CSRC\_A), q)$
  - $ixcol = \text{mod}(((jx-1)/NB\_X)+CSRC\_X), q)$
3. If  $incx = 1 (\neq M\_X)$ , then (in the process grid) the process row containing the first row of the submatrix  $A$  does not contain the first row of the submatrix  $X$ ; that is,  $iarow \neq ixrow$ , where:
  - $iarow = \text{mod}((((ia-1)/MB\_A)+RSRC\_A), p)$

- $ixrow = \text{mod}(((ix-1)/MB\_X)+RSRC\_X), p)$
- 4.  $LLD\_A < \max(1, LOCp(M\_A))$
- 5.  $LLD\_X < \max(1, LOCp(M\_X))$

## Examples

### Example 1

This example computes  $A = \alpha x x^T + A$  using a  $2 \times 2$  process grid.

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      UPLO  N  ALPHA  X  IX  JX  DESC_X  INCX  A  IA  JA  DESC_A
      |    |    |    |  |  |  |    |    |  |  |  |
CALL PDSYR( 'L' , 9 , 1.0D0 , X , 1 , 1 , DESC_X , 1 , A , 1 , 1 , DESC_A)
```

	Desc_A	Desc_X
DTYPE_	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	9	9
N_	9	1
MB_	4	4
NB_	4	1
RSRC_	0	0
CSRC_	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>

#### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:  
 $LLD\_A = \max(1, \text{NUMROC}(M\_A, MB\_A, MYROW, RSRC\_A, NPROW))$   
 $LLD\_X = \max(1, \text{NUMROC}(M\_X, MB\_X, MYROW, RSRC\_X, NPROW))$   
 In this example,  $LLD\_A = 5$  on  $P_{00}$  and  $P_{01}$ ,  $LLD\_A = 4$  on  $P_{10}$  and  $P_{11}$ ,  
 $LLD\_X = 5$  on  $P_{00}$ , and  $LLD\_X = 4$  on  $P_{10}$ .

Global real symmetric matrix  $A$  of order 9 with block size  $4 \times 4$ :

B,D	0	1	2
0	$\begin{bmatrix} 1.0 & . & . & . \\ 2.0 & 12.0 & . & . \\ 3.0 & 13.0 & 23.0 & . \\ 4.0 & 14.0 & 24.0 & 34.0 \end{bmatrix}$	$\begin{bmatrix} . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \end{bmatrix}$	$\begin{bmatrix} . \\ . \\ . \\ . \end{bmatrix}$
1	$\begin{bmatrix} 5.0 & 15.0 & 25.0 & 35.0 \\ 6.0 & 16.0 & 26.0 & 36.0 \\ 7.0 & 17.0 & 27.0 & 37.0 \\ 8.0 & 18.0 & 28.0 & 38.0 \end{bmatrix}$	$\begin{bmatrix} 45.0 & . & . & . \\ 46.0 & 56.0 & . & . \\ 47.0 & 57.0 & 67.0 & . \\ 48.0 & 58.0 & 68.0 & 78.0 \end{bmatrix}$	$\begin{bmatrix} . \\ . \\ . \\ . \end{bmatrix}$

$$2 \left[ \begin{array}{c|c|c} \hline & & \\ \hline 9.0 & 19.0 & 29.0 & 39.0 & \hline & 49.0 & 59.0 & 69.0 & 79.0 & \hline & & & & & 89.0 & \hline \end{array} \right]$$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

Local arrays for A:

p,q	0	1
0	<div> <div>1.0</div> <div>2.0 12.0</div> <div>3.0 13.0 23.0</div> <div>4.0 14.0 24.0 34.0</div> <div>5.0 15.0 25.0 35.0</div> </div>	<div> <div>49.0 59.0 69.0 79.0</div> </div>
1	<div> <div>6.0 16.0 26.0 36.0</div> <div>7.0 17.0 27.0 37.0</div> <div>8.0 18.0 28.0 38.0</div> </div>	<div> <div>45.0 55.0 65.0 75.0</div> <div>46.0 56.0 66.0 76.0</div> <div>47.0 57.0 67.0 77.0</div> <div>48.0 58.0 68.0 78.0</div> </div>

Global vector  $x$  of size  $9 \times 1$  with block size 4:

$$B,D \quad 0 \quad \left[ \begin{array}{c} 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \\ \hline 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \\ \hline 1.0 \end{array} \right]$$

The following is the  $2 \times 2$  process grid:

B,D	0	--
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

Local arrays for  $x$ :

$$p,q \quad 0 \quad \left[ \begin{array}{c} 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \\ \hline 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \end{array} \right]$$

## PDSYR and PZHER

### Output:

Global real symmetric matrix  $A$  of order 9 with block size  $4 \times 4$ :

B,D	0	1	2
0	<div> <div>2.0 . . .</div> <div>3.0 13.0 . .</div> <div>4.0 14.0 24.0 .</div> <div>5.0 15.0 25.0 35.0</div> </div>	<div> <div>. . . .</div> <div>. . . .</div> <div>. . . .</div> <div>. . . .</div> </div>	<div> <div>.</div> <div>.</div> <div>.</div> <div>.</div> </div>
1	<div> <div>6.0 16.0 26.0 36.0</div> <div>7.0 17.0 27.0 37.0</div> <div>8.0 18.0 28.0 38.0</div> <div>9.0 19.0 29.0 39.0</div> </div>	<div> <div>46.0 . . .</div> <div>47.0 57.0 . .</div> <div>48.0 58.0 68.0 .</div> <div>49.0 59.0 69.0 79.0</div> </div>	<div> <div>.</div> <div>.</div> <div>.</div> <div>.</div> </div>
2	<div> <div>10.0 20.0 30.0 40.0</div> </div>	<div> <div>50.0 60.0 70.0 80.0</div> </div>	<div> <div>90.0</div> </div>

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$

Local arrays for  $A$ :

p,q	0	1
0	<div> <div>2.0 . . . .</div> <div>3.0 13.0 . . .</div> <div>4.0 14.0 24.0 . .</div> <div>5.0 15.0 25.0 35.0 .</div> <div>10.0 20.0 30.0 40.0 90.0</div> </div>	<div> <div>. . . .</div> <div>. . . .</div> <div>. . . .</div> <div>. . . .</div> <div>50.0 60.0 70.0 80.0</div> </div>
1	<div> <div>6.0 16.0 26.0 36.0 .</div> <div>7.0 17.0 27.0 37.0 .</div> <div>8.0 18.0 28.0 38.0 .</div> <div>9.0 19.0 29.0 39.0 .</div> </div>	<div> <div>46.0 . . .</div> <div>47.0 57.0 . .</div> <div>48.0 58.0 68.0 .</div> <div>49.0 59.0 69.0 79.0</div> </div>

### Example 2

This example computes  $A = \alpha x x^H + A$  using a  $2 \times 2$  process grid.

**Note:** The imaginary parts of the diagonal elements of a complex Hermitian matrix are assumed to be zero, so you do not have to set these values. On output, they are set to zero except when  $N$  is zero or  $\alpha$  is zero.

### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      UPLO N  ALPHA  X  IX  JX  DESC_X  INCX  A  IA  JA  DESC_A
      |   |   |     |  |  |  |       |   |  |  |  |
CALL PZHER( 'L' , 3 , 1.0D0 , X , 1 , 1 , DESC_X , 1 , A , 1 , 1 , DESC_A)
```

	Desc_A	Desc_X
DTYPE_	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	3	3
N_	3	1
MB_	2	2
NB_	2	1
RSRC_	0	0
CSRC_	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.

2. Each process should set the LLD\_ as follows:

$$\text{LLD\_A} = \text{MAX}(1, \text{NUMROC}(\text{M\_A}, \text{MB\_A}, \text{MYROW}, \text{RSRC\_A}, \text{NPROW}))$$

$$\text{LLD\_X} = \text{MAX}(1, \text{NUMROC}(\text{M\_X}, \text{MB\_X}, \text{MYROW}, \text{RSRC\_X}, \text{NPROW}))$$

In this example, LLD\_A = 2 on P<sub>00</sub> and P<sub>01</sub>, LLD\_A = 1 on P<sub>10</sub> and P<sub>11</sub>,

LLD\_X = 2 on P<sub>00</sub>, and LLD\_X = 1 on P<sub>10</sub>.

Global complex Hermitian matrix *A* of order 3 with block size 2 × 2:

$$\begin{array}{c} \text{B,D} \end{array} \quad \begin{array}{cc} 0 & 1 \end{array}$$

$$\begin{array}{c} 0 \\ 1 \end{array} \quad \left[ \begin{array}{cc|c} (1.0, 0.0) & . & . \\ (3.0, -5.0) & (7.0, 0.0) & . \\ \hline (2.0, 3.0) & (4.0, 8.0) & (6.0, 0.0) \end{array} \right]$$

The following is the 2 × 2 process grid:

$$\begin{array}{c} \text{B,D} \end{array} \quad \begin{array}{c|c} 0 & 2 \\ \hline 0 & P_{00} \\ \hline 1 & P_{10} \end{array} \quad \begin{array}{c|c} 1 & \\ \hline & P_{01} \\ \hline & P_{11} \end{array}$$

Local arrays for *A*:

$$\begin{array}{c} \text{p,q} \end{array} \quad \begin{array}{cc} 0 & 1 \end{array}$$

$$\begin{array}{c} 0 \\ 1 \end{array} \quad \left[ \begin{array}{cc|c} (1.0, .) & . & . \\ (3.0, -5.0) & (7.0, .) & . \\ \hline (2.0, 3.0) & (4.0, 8.0) & (6.0, .) \end{array} \right]$$

Global vector *x* of size 3 × 1 with block size 2:

$$\begin{array}{c} \text{B,D} \end{array} \quad 0$$

$$\begin{array}{c} 0 \\ 1 \end{array} \quad \left[ \begin{array}{c} (1.0, 2.0) \\ (4.0, 0.0) \\ \hline (3.0, 4.0) \end{array} \right]$$

The following is the 2 × 2 process grid:

## PDSYR and PZHER

B,D	0	--
0	P <sub>00</sub>	P <sub>01</sub>
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for  $x$ :

p,q	0
0	(1.0,2.0) (4.0,0.0)
1	(3.0,4.0)

**Output:**

Global complex Hermitian matrix  $A$  of order 3 with block size  $2 \times 2$ :

B,D	0	1
0	( 6.0, 0.0) ( 7.0,-13.0) (23.0, 0.0)	$\cdot$ $\cdot$
1	(13.0, 1.0) (16.0,24.0)	(31.0, 0.0)

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	P <sub>00</sub>	P <sub>01</sub>
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for  $A$ :

p,q	0	1
0	( 6.0, 0.0) ( 7.0,-13.0) (23.0, 0.0)	$\cdot$ $\cdot$
1	(13.0, 1.0) (16.0,24.0)	(31.0, 0.0)

## PDSYR2 and PZHER2 — Rank-Two Update of a Real Symmetric or a Complex Hermitian Matrix

### Purpose

PDSYR2 computes the following rank-two update:

$$\bullet \ A \leftarrow \alpha xy^T + \alpha yx^T + A$$

PZHER2 computes the following rank-two update:

$$\mathbf{A} \leftarrow \alpha \mathbf{xy}^H + \overline{\alpha} \mathbf{yx}^H + \mathbf{A}$$

where, in the formula above:

- $A$  represents the global submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$ .
- $x$  represents the global vector:
  - For  $incx = M\_X$ , it is  $X_{ix:ix, jx:jx+n-1}$ .
  - For  $incx = 1$  and  $incx \neq M\_X$ , it is  $X_{ix:ix+n-1, jx:jx}$ .
- $y$  represents the global vector:
  - For  $incy = M\_Y$ , it is  $Y_{iy:iy, jy:jy+n-1}$ .
  - For  $incy = 1$  and  $incy \neq M\_Y$ , it is  $Y_{iy:iy+n-1, jy:jy}$ .
- $\alpha$  is a scalar.

and:

- For PDSYR2, submatrix  $A$  is real symmetric.
- For PZHER2, submatrix  $A$  is complex Hermitian.

**Note:** No data should be moved to form  $x^T$ ,  $x^H$ ,  $y^T$ , or  $y^H$ ; that is, the vectors  $x$  and  $y$  should always be stored in their untransposed form.

In the following two cases, no computation is performed and the subroutine returns after doing some parameter checking:

- $n = 0$
- $\alpha$  is zero.

See references [15] and [16].

Table 53. Data Types

$A, x, y, \alpha$	Subprogram
Long-precision real	PDSYR2
Long-precision complex	PZHER2

### Syntax

<b>Fortran</b>	CALL PDSYR2   PZHER2 ( <i>uplo</i> , <i>n</i> , <i>alpha</i> , <i>x</i> , <i>ix</i> , <i>jx</i> , <i>desc_x</i> , <i>incx</i> , <i>y</i> , <i>iy</i> , <i>jy</i> , <i>desc_y</i> , <i>incy</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> )
<b>C and C++</b>	pdsyr2   pzher2 ( <i>uplo</i> , <i>n</i> , <i>alpha</i> , <i>x</i> , <i>ix</i> , <i>jx</i> , <i>desc_x</i> , <i>incx</i> , <i>y</i> , <i>iy</i> , <i>jy</i> , <i>desc_y</i> , <i>incy</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> );

### On Entry

*uplo* indicates whether the upper or lower triangular part of the global symmetric submatrix  $A$  is referenced, where:

If *uplo* = 'U', the upper triangular part is referenced.

If *uplo* = 'L', the lower triangular part is referenced.

Scope: **global**

Specified as: a single character;  $uplo = 'U'$  or  $'L'$ .

$n$  is the number of rows and columns in submatrix  $A$  and the number of elements in vectors  $x$  and  $y$  used in the computation.

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

$alpha$  is the scalar  $\alpha$ .

Scope: **global**

Specified as: a number of the data type indicated in Table 53 on page 197.

$x$  is the local part of the global matrix  $X$ . This identifies the **first element** of the local array  $X$ . This subroutine computes the location of the first element of the local subarray used, based on  $ix$ ,  $jx$ ,  $desc\_x$ ,  $p$ ,  $q$ ,  $myrow$ , and  $mycol$ ; therefore:

- If  $incx = M\_X$ , the leading  $LOCp(ix)$  by  $LOCq(jx+n-1)$  part of the local array  $X$  must contain the local pieces of the leading  $ix$  by  $jx+n-1$  part of the global matrix.
- If  $incx = 1$  and  $incx \neq M\_X$ , the leading  $LOCp(ix+n-1)$  by  $LOCq(jx)$  part of the local array  $X$  must contain the local pieces of the leading  $ix+n-1$  by  $jx$  part of the global matrix.

**Note:** No data should be moved to form  $x^T$  or  $x^H$ ; that is, the vector  $x$  should always be stored in its untransposed form.

Scope: **local**

Specified as: an  $LLD\_X$  by (at least)  $LOCq(N\_X)$  array, containing numbers of the data type indicated in Table 53 on page 197. Details about the block-cyclic data distribution of the global matrix  $X$  are stored in  $desc\_x$ .

$ix$  has the following meaning:

If  $incx = M\_X$ , it indicates which row of global matrix  $X$  is used for vector  $x$ .

If  $incx = 1$  and  $incx \neq M\_X$ , it is the row index of global matrix  $X$ , identifying the first element of vector  $x$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ix \leq M\_X$  and:

If  $incx = 1$  and  $incx \neq M\_X$ , then  $ix+n-1 \leq M\_X$ .

$jx$  has the following meaning:

If  $incx = M\_X$ , it is the column index of global matrix  $X$ , identifying the first element of vector  $x$ .

If  $incx = 1$  and  $incx \neq M\_X$ , it indicates which column of global matrix  $X$  is used for vector  $x$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq jx \leq N\_X$  and:

If  $incx = M\_X$ , then  $jx+n-1 \leq N\_X$ .

$desc\_x$  is the array descriptor for global matrix  $X$ , described in the following table:



<i>desc_x</i>	Name	Description	Limits	Scope
1	DTYPE_X	Descriptor type	DTYPE_X=1	Global
2	CTXT_X	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_X	Number of rows in the global matrix	If $n = 0$ : $M\_X \geq 0$ Otherwise: $M\_X \geq 1$	Global
4	N_X	Number of columns in the global matrix	If $n = 0$ : $N\_X \geq 0$ Otherwise: $N\_X \geq 1$	Global
5	MB_X	Row block size	$MB\_X \geq 1$	Global
6	NB_X	Column block size	$NB\_X \geq 1$	Global
7	RSRC_X	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_X < p$	Global
8	CSRC_X	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_X < q$	Global
9	LLD_X	The leading dimension of the local array	$LLD\_X \geq \max(1, LOCp(M\_X))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

*incx* is the stride for global vector  $x$ .

Scope: **global**

Specified as: a fullword integer;  $incx = 1$  or  $incx = M\_X$ , where:

If  $incx = M\_X$ , then  $x$  is a row-distributed vector.

If  $incx = 1$  and  $incx \neq M\_X$ , then  $x$  is a column-distributed vector.

$y$  is the local part of the global matrix  $Y$ . This identifies the **first element** of the local array  $Y$ . This subroutine computes the location of the first element of the local subarray used, based on  $iy$ ,  $jy$ ,  $desc_y$ ,  $p$ ,  $q$ ,  $myrow$ , and  $mycol$ ; therefore:

- If  $incy = M\_Y$ , the leading  $LOCp(iy)$  by  $LOCq(jy+n-1)$  part of the local array  $Y$  must contain the local pieces of the leading  $iy$  by  $jy+n-1$  part of the global matrix.
- If  $incy = 1$  and  $incy \neq M\_Y$ , the leading  $LOCp(iy+n-1)$  by  $LOCq(jy)$  part of the local array  $Y$  must contain the local pieces of the leading  $iy+n-1$  by  $jy$  part of the global matrix.

**Note:** No data should be moved to form  $y^T$  or  $y^H$ ; that is, the vector  $x$  should always be stored in its untransposed form.

Scope: **local**

Specified as: an  $LLD\_Y$  by (at least)  $LOCq(N\_Y)$  array, containing numbers of the data type indicated in Table 53 on page 197. Details about the block-cyclic data distribution of the global matrix  $Y$  are stored in  $desc_y$ .

*iy* has the following meaning:  
If *incy* = *M\_Y*, it indicates which row of global matrix *Y* is used for vector *y*.

If *incy* = 1 and *incy* ≠ *M\_Y*, it is the row index of global matrix *Y*, identifying the first element of vector *y*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq iy \leq M\_Y$  and:

If *incy* = 1 and *incy* ≠ *M\_Y*, then  $iy+n-1 \leq M\_Y$ .

*jy* has the following meaning:

If *incy* = *M\_Y*, it is the column index of global matrix *Y*, identifying the first element of vector *y*.

If *incy* = 1 and *incy* ≠ *M\_Y*, it indicates which column of global matrix *Y* is used for vector *y*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq jy \leq N\_Y$  and:

If *incy* = *M\_Y*, then  $jy+n-1 \leq N\_Y$ .

*desc\_y* is the array descriptor for global matrix *Y*, described in the following table:

<i>desc_y</i>	Name	Description	Limits	Scope
1	DTYPE_Y	Descriptor type	DTYPE_Y=1	Global
2	CTXT_Y	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_Y	Number of rows in the global matrix	If $n = 0$ : $M\_Y \geq 0$ Otherwise: $M\_Y \geq 1$	Global
4	N_Y	Number of columns in the global matrix	If $n = 0$ : $N\_Y \geq 0$ Otherwise: $N\_Y \geq 1$	Global
5	MB_Y	Row block size	$MB\_Y \geq 1$	Global
6	NB_Y	Column block size	$NB\_Y \geq 1$	Global
7	RSRC_Y	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_Y < p$	Global
8	CSRC_Y	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_Y < q$	Global
9	LLD_Y	The leading dimension of the local array	$LLD\_Y \geq \max(1, LOCP(M\_Y))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

*incy* is the stride for global vector *y*.

Scope: **global**

Specified as: a fullword integer;  $incy = 1$  or  $incy = M\_X$ , where:

If  $incy = M\_Y$ , then  $y$  is a row-distributed vector.

If  $incy = 1$  and  $incy \neq M\_Y$ , then  $y$  is a column-distributed vector.

$a$  is the local part of the global real symmetric or complex Hermitian matrix  $A$ . This identifies the **first element** of the local array  $A$ . This subroutine computes the location of the first element of the local subarray used, based on  $ia$ ,  $ja$ ,  $desc\_a$ ,  $p$ ,  $q$ ,  $myrow$ , and  $mycol$ ; therefore, the leading  $LOCp(ia+n-1)$  by  $LOCq(ja+n-1)$  part of the local array  $A$  must contain the local pieces of the leading  $ia+n-1$  by  $ja+n-1$  part of the global matrix, and:

- If  $uplo = 'U'$ , the leading  $n \times n$  upper triangular part of the global symmetric submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  must contain the upper triangular part of the submatrix, and the strictly lower triangular part is not referenced.
- If  $uplo = 'L'$ , the leading  $n \times n$  lower triangular part of the global symmetric submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  must contain the lower triangular part of the submatrix, and the strictly upper triangular part is not referenced.

Scope: **local**

Specified as: an  $LLD\_A$  by (at least)  $LOCq(N\_A)$  array, containing numbers of the data type indicated in Table 53 on page 197. Details about the square block-cyclic data distribution of global matrix  $A$  are stored in  $desc\_a$ .

$ia$  is the row index of the global matrix  $A$ , identifying the first row of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq M\_A$  and  $ia+n-1 \leq M\_A$ .

$ja$  is the column index of the global matrix  $A$ , identifying the first column of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$  and  $ja+n-1 \leq N\_A$ .

$desc\_a$  is the array descriptor for global matrix  $A$ , described in the following table:

$desc\_a$	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global

<i>desc_a</i>	Name	Description	Limits	Scope
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq \text{RSRC\_A} < p$	Global
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq \text{CSRC\_A} < q$	Global
9	LLD_A	The leading dimension of the local array	$\text{LLD\_A} \geq \max(1, \text{LOCp}(\text{M\_A}))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

### On Return

*a* is the updated local part of the global matrix *A*, containing the results of the computation.

Scope: **local**

Returned as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 53 on page 197.

### Notes and Coding Rules

- These subroutines accept lowercase letters for the *uplo* argument.
- The matrix and vectors must have no common elements; otherwise, results are unpredictable.
- The imaginary parts of the diagonal elements of the complex Hermitian matrix are assumed to be zero, so you do not have to set these values. On output, they are set to zero except when N is zero or  $\alpha$  is zero, in which case no computation is performed.
- The NUMROC utility subroutine can be used to determine the values of LOCp(M\_) and LOCq(N\_) used in the argument descriptions above. For details, see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28 and “NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process” on page 868.
- For suggested block sizes, see “Coding Tips for Optimizing Parallel Performance” on page 77.
- The following values must be equal: CTXT\_A = CTXT\_X = CTXT\_Y.
- The vectors *x* and *y* must be distributed along the same axis—that is, they must both be row distributed or column distributed, where:
  - $\text{incx} = \text{M\_X}$  and  $\text{incy} = \text{M\_Y}$  for row distribution
  - $\text{incx} = 1$  ( $\neq \text{M\_X}$ ) and  $\text{incy} = 1$  ( $\neq \text{M\_Y}$ ) for column distribution
- The global symmetric matrix *A* must be distributed using a square block-cyclic distribution; that is, MB\_A = NB\_A.
- The block row and block column offsets of the global symmetric matrix *A* must be equal; that is,  $\text{mod}(ia-1, \text{MB\_A}) = \text{mod}(ja-1, \text{NB\_A})$ .
- If  $\text{incx} = \text{M\_X}$ :
  - In the process grid, the process column containing the first column of the submatrix *X* must also contain the first column of the submatrix *A*; that is,  $\text{iacol} = \text{ixcol}$ , where:
    - $\text{iacol} = \text{mod}(((ja-1)/\text{NB\_A}) + \text{CSRC\_A}), q)$

- $ixcol = \text{mod}(((jx-1)/NB\_X)+CSRC\_X), q)$
  - The block column offset of  $x$  must be equal to the block row offset of  $A$ ; that is,  $\text{mod}(jx-1, NB\_X) = \text{mod}(ia-1, MB\_A)$ .
  - The following block sizes must be equal:  $NB\_X = NB\_A$ .
11. If  $incx = 1$  ( $\neq M\_X$ ):
- In the process grid, the process row containing the first row of the submatrix  $X$  must also contain the first row of the submatrix  $A$ ; that is,  $iarow = ixrow$ , where:
    - $iarow = \text{mod}(((ia-1)/MB\_A)+RSRC\_A), p)$
    - $ixrow = \text{mod}(((ix-1)/MB\_X)+RSRC\_X), p)$
  - The block row offset of  $x$  must be equal to the block row offset of  $A$ ; that is,  $\text{mod}(ix-1, MB\_X) = \text{mod}(ia-1, MB\_A)$ .
  - The following block sizes must be equal:  $MB\_X = MB\_A$ .
12. If  $incy = M\_Y$ :
- In the process grid, the process column containing the first column of the submatrix  $Y$  must also contain the first column of the submatrix  $A$ ; that is,  $iacol = iycol$ , where:
    - $iacol = \text{mod}(((ja-1)/NB\_A)+CSRC\_A), q)$
    - $iycol = \text{mod}(((jy-1)/NB\_Y)+CSRC\_Y), q)$
  - The block column offset of  $y$  must be equal to the block row offset of  $A$ ; that is,  $\text{mod}(jy-1, NB\_Y) = \text{mod}(ia-1, MB\_A)$ .
  - The following block sizes must be equal:  $NB\_Y = NB\_A$ .
13. If  $incy = 1$  ( $\neq M\_Y$ ):
- In the process grid, the process row containing the first row of the submatrix  $Y$  must also contain the first row of the submatrix  $A$ ; that is,  $iarow = iyrow$ , where:
    - $iarow = \text{mod}(((ia-1)/MB\_A)+RSRC\_A), p)$
    - $iyrow = \text{mod}(((iy-1)/MB\_Y)+RSRC\_Y), p)$
  - The block row offset of  $y$  must be equal to the block row offset of  $A$ ; that is,  $\text{mod}(iy-1, MB\_Y) = \text{mod}(ia-1, MB\_A)$ .
  - The following block sizes must be equal:  $MB\_Y = MB\_A$ .

## Error Conditions

### Computational Errors

None

### Resource Errors

Unable to allocate work space

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1.  $DTYPE\_A$  is invalid.
2.  $DTYPE\_X$  is invalid.
3.  $DTYPE\_Y$  is invalid.

#### Stage 2:

1.  $CTXT\_A$  is invalid.

#### Stage 3:

1. This subroutine was called from outside the process grid.

**Stage 4:**

1.  $uplo \neq 'U' \text{ or } 'L'$
2.  $n < 0$
3.  $M\_X < 0 \text{ and } n = 0; M\_X < 1 \text{ otherwise}$
4.  $N\_X < 0 \text{ and } n = 0; N\_X < 1 \text{ otherwise}$
5.  $MB\_X < 1$
6.  $NB\_X < 1$
7.  $RSRC\_X < 0 \text{ or } RSRC\_X \geq p$
8.  $CSRC\_X < 0 \text{ or } CSRC\_X \geq q$
9.  $CTXT\_A \neq CTXT\_X$
10.  $ix < 1$
11.  $jx < 1$
12.  $M\_Y < 0 \text{ and } n = 0; M\_Y < 1 \text{ otherwise}$
13.  $N\_Y < 0 \text{ and } n = 0; N\_Y < 1 \text{ otherwise}$
14.  $MB\_Y < 1$
15.  $NB\_Y < 1$
16.  $RSRC\_Y < 0 \text{ or } RSRC\_Y \geq p$
17.  $CSRC\_Y < 0 \text{ or } CSRC\_Y \geq q$
18.  $CTXT\_A \neq CTXT\_Y$
19.  $iy < 1$
20.  $jy < 1$
21.  $M\_A < 0 \text{ and } n = 0; M\_A < 1 \text{ otherwise}$
22.  $N\_A < 0 \text{ and } n = 0; N\_A < 1 \text{ otherwise}$
23.  $MB\_A < 1$
24.  $NB\_A < 1$
25.  $RSRC\_A < 0 \text{ or } RSRC\_A \geq p$
26.  $CSRC\_A < 0 \text{ or } CSRC\_A \geq q$
27.  $ia < 1$
28.  $ja < 1$

**Stage 5:**

1.  $NB\_A \neq MB\_A$   
If  $n \neq 0$ :
2.  $ia > M\_A$
3.  $ja > N\_A$
4.  $ia+n-1 > M\_A$
5.  $ja+n-1 > N\_A$
6.  $ix > M\_X$
7.  $jx > N\_X$
8.  $iy > M\_Y$
9.  $jy > N\_Y$   
If  $incx = M\_X$ :
10.  $NB\_X \neq NB\_A$
11.  $\text{mod}(jx-1, NB\_X) \neq \text{mod}(ia-1, MB\_A)$
12.  $n \neq 0 \text{ and } jx+n-1 > N\_X$   
If  $incx = 1 (\neq M\_X)$ :
13.  $MB\_X \neq MB\_A$
14.  $\text{mod}(ix-1, MB\_X) \neq \text{mod}(ia-1, MB\_A)$
15.  $n \neq 0 \text{ and } ix+n-1 > M\_X$   
Otherwise:
16.  $incx \neq M\_X \text{ and } incx \neq 1$   
If  $incy = M\_Y$ :
17.  $NB\_Y \neq NB\_A$
18.  $\text{mod}(jy-1, NB\_Y) \neq \text{mod}(ia-1, MB\_A)$
19.  $n \neq 0 \text{ and } jy+n-1 > N\_Y$

- If  $incy = 1$  ( $\neq M\_Y$ ):
20.  $MB\_Y \neq MB\_A$
  21.  $\text{mod}(iy-1, MB\_Y) \neq \text{mod}(ia-1, MB\_A)$
  22.  $n \neq 0$  and  $iy+n-1 > M\_Y$
- Otherwise:
23.  $incy \neq M\_Y$  and  $incy \neq 1$

**Stage 6:**

1.  $\text{mod}(ja-1, NB\_A) \neq \text{mod}(ia-1, MB\_A)$
2. If  $incx = M\_X$ , then (in the process grid) the process column containing the first column of the submatrix  $A$  does not contain the first column of the submatrix  $X$ ; that is,  $iacol \neq ixcol$ , where:
  - $iacol = \text{mod}(((ja-1)/NB\_A)+CSRC\_A), q)$
  - $ixcol = \text{mod}(((jx-1)/NB\_X)+CSRC\_X), q)$
3. If  $incx = 1$  ( $\neq M\_X$ ), then (in the process grid) the process row containing the first row of the submatrix  $A$  does not contain the first row of the submatrix  $X$ ; that is,  $iarow \neq ixrow$ , where:
  - $iarow = \text{mod}(((ia-1)/MB\_A)+RSRC\_A), p)$
  - $ixrow = \text{mod}(((ix-1)/MB\_X)+RSRC\_X), p)$
4. If  $incy = M\_Y$ , then (in the process grid) the process column containing the first column of the submatrix  $A$  does not contain the first column of the submatrix  $Y$ ; that is,  $iacol \neq iycol$ , where:
  - $iacol = \text{mod}(((ja-1)/NB\_A)+CSRC\_A), q)$
  - $iycol = \text{mod}(((jy-1)/NB\_Y)+CSRC\_Y), q)$
5. If  $incy = 1$  ( $\neq M\_Y$ ), then (in the process grid) the process row containing the first row of the submatrix  $A$  does not contain the first row of the submatrix  $Y$ ; that is,  $iarow \neq iyrow$ , where:
  - $iarow = \text{mod}(((ia-1)/MB\_A)+RSRC\_A), p)$
  - $iyrow = \text{mod}(((iy-1)/MB\_Y)+RSRC\_Y), p)$
6.  $LLD\_A < \max(1, LOCp(M\_A))$
7.  $LLD\_X < \max(1, LOCp(M\_X))$
8.  $LLD\_Y < \max(1, LOCp(M\_Y))$

## Examples

### Example 1

This example computes  $A = \alpha xy^T + \alpha yx^T + A$  using a  $2 \times 2$  process grid.

**Call Statements and Input:**

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      UPLO  N    ALPHA  X  IX  JX    DESC_X  INCX  Y  IY  JY
      |    |    |      |  |  |    |      |    |  |  |  |
CALL PDSYR2( 'L' , 9 , 1.0D0 , X , 1 , 1 , DESC_X , 1 , Y , 1 , 1 ,

      DESC_Y  INCY  A  IA  JA  DESC_A
      |      |    |  |  |    |
      DESC_Y , 1 , A , 1 , 1 , DESC_A )
```

	Desc_A	Desc_X	Desc_Y
DTYPE_	1	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>

	Desc_A	Desc_X	Desc_Y
M_	9	9	9
N_	9	1	1
MB_	4	4	4
NB_	4	1	1
RSRC_	0	0	0
CSRC_	0	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>	See below <sup>2</sup>

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:  
 $LLD\_A = \text{MAX}(1, \text{NUMROC}(M\_A, MB\_A, MYROW, RSRC\_A, NPROW))$   
 $LLD\_X = \text{MAX}(1, \text{NUMROC}(M\_X, MB\_X, MYROW, RSRC\_X, NPROW))$   
 $LLD\_Y = \text{MAX}(1, \text{NUMROC}(M\_Y, MB\_Y, MYROW, RSRC\_Y, NPROW))$   
 In this example,  $LLD\_A = 5$  on  $P_{00}$  and  $P_{01}$ ,  $LLD\_A = 4$  on  $P_{10}$  and  $P_{11}$ ,  
 $LLD\_X = LLD\_Y = 5$  on  $P_{00}$ , and  $LLD\_X = LLD\_Y = 4$  on  $P_{10}$ .

Global real symmetric matrix  $A$  of order 9 with block size  $4 \times 4$ :

B,D	0	1	2
0	$\begin{bmatrix} 1.0 & . & . & . \\ 2.0 & 12.0 & . & . \\ 3.0 & 13.0 & 23.0 & . \\ 4.0 & 14.0 & 24.0 & 34.0 \end{bmatrix}$	$\begin{bmatrix} . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \end{bmatrix}$	$\begin{bmatrix} . \\ . \\ . \\ . \end{bmatrix}$
1	$\begin{bmatrix} 5.0 & 15.0 & 25.0 & 35.0 \\ 6.0 & 16.0 & 26.0 & 36.0 \\ 7.0 & 17.0 & 27.0 & 37.0 \\ 8.0 & 18.0 & 28.0 & 38.0 \end{bmatrix}$	$\begin{bmatrix} 45.0 & . & . & . \\ 46.0 & 56.0 & . & . \\ 47.0 & 57.0 & 67.0 & . \\ 48.0 & 58.0 & 68.0 & 78.0 \end{bmatrix}$	$\begin{bmatrix} . \\ . \\ . \\ . \end{bmatrix}$
2	$\begin{bmatrix} 9.0 & 19.0 & 29.0 & 39.0 \end{bmatrix}$	$\begin{bmatrix} 49.0 & 59.0 & 69.0 & 79.0 \end{bmatrix}$	$\begin{bmatrix} 89.0 \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

Local arrays for  $A$ :

p,q	0	1
0	$\begin{bmatrix} 1.0 & . & . & . \\ 2.0 & 12.0 & . & . \\ 3.0 & 13.0 & 23.0 & . \\ 4.0 & 14.0 & 24.0 & 34.0 \\ 9.0 & 19.0 & 29.0 & 39.0 & 89.0 \end{bmatrix}$	$\begin{bmatrix} . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ 49.0 & 59.0 & 69.0 & 79.0 \end{bmatrix}$
1	$\begin{bmatrix} 5.0 & 15.0 & 25.0 & 35.0 \\ 6.0 & 16.0 & 26.0 & 36.0 \\ 7.0 & 17.0 & 27.0 & 37.0 \\ 8.0 & 18.0 & 28.0 & 38.0 \end{bmatrix}$	$\begin{bmatrix} 45.0 & . & . & . \\ 46.0 & 56.0 & . & . \\ 47.0 & 57.0 & 67.0 & . \\ 48.0 & 58.0 & 68.0 & 78.0 \end{bmatrix}$

Global vector  $x$  of size  $9 \times 1$  with block size 4:



B,D	0
	1.0
	1.0
0	1.0
	1.0
	----
	1.0
	1.0
1	1.0
	1.0
	----
2	1.0

The following is the  $2 \times 2$  process grid:

B,D	0	--
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for  $x$ :

p,q	0
	----
	1.0
	1.0
0	1.0
	1.0
	1.0
	----
	1.0
	1.0
1	1.0
	1.0

Global vector  $y$  of size  $9 \times 1$  with block size 4:

B,D	0
	2.0
	2.0
0	2.0
	2.0
	----
	2.0
	2.0
1	2.0
	2.0
	----
2	2.0

The following is the  $2 \times 2$  process grid:

B,D	0	--
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for  $y$ :

## PDSYR2 and PZHER2

p,q	0
0	2.0
	2.0
	2.0
	2.0
	2.0
1	2.0
	2.0
	2.0
	2.0

### Output:

Global real symmetric matrix  $A$  of order 9 with block size  $4 \times 4$ :

B,D	0	1	2
0	5.0 . . .	. . . .	.
	6.0 16.0 . .	. . . .	.
	7.0 17.0 27.0 .	. . . .	.
	8.0 18.0 28.0 38.0	. . . .	.
1	9.0 19.0 29.0 39.0	49.0 . . .	.
	10.0 20.0 30.0 40.0	50.0 60.0 . .	.
	11.0 21.0 31.0 41.0	51.0 61.0 71.0 .	.
	12.0 22.0 32.0 42.0	52.0 62.0 72.0 82.0	.
2	13.0 23.0 33.0 43.0	53.0 63.0 73.0 83.0	93.0

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

Local arrays for  $A$ :

p,q	0	1
0	5.0 . . . .	. . . .
	6.0 16.0 . . .	. . . .
	7.0 17.0 27.0 . .	. . . .
	8.0 18.0 28.0 38.0 .	. . . .
	13.0 23.0 33.0 43.0 93.0	53.0 63.0 73.0 83.0
1	9.0 19.0 29.0 39.0 .	49.0 . . .
	10.0 20.0 30.0 40.0 .	50.0 60.0 . .
	11.0 21.0 31.0 41.0 .	51.0 61.0 71.0 .
	12.0 22.0 32.0 42.0 .	52.0 62.0 72.0 82.0

### Example 2

This example computes:

$$A \leftarrow A + \alpha xy^H + \bar{\alpha} yx^H$$

using a  $2 \times 2$  process grid.

**Note:** The imaginary parts of the diagonal elements of a complex Hermitian matrix are assumed to be zero, so you do not have to set these values. On output, they are set to zero except when N is zero or  $\alpha$  is zero.

**Call Statements and Input:**

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      UPLO  N    ALPHA  X  IX  JX  DESC_X  INCX  Y  IY  JY
      |    |    |      |  |  |  |      |  |   |  |  |
CALL PZHER2( 'L' , 3 , ALPHA , X , 1 , 1 , DESC_X , 1 , Y , 1 , 1 ,

      DESC_Y  INCY  A  IA  JA  DESC_A
      |      |    |  |  |  |
      DESC_Y , 1 , A , 1 , 1 , DESC_A )

ALPHA = (1.0,0.0)
```

	Desc_A	Desc_X	Desc_Y
DTYPE_	1	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	3	3	3
N_	3	1	1
MB_	2	2	2
NB_	2	1	1
RSRC_	0	0	0
CSRC_	0	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>	See below <sup>2</sup>

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:

```
LLD_A = MAX(1,NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_X = MAX(1,NUMROC(M_X, MB_X, MYROW, RSRC_X, NPROW))
LLD_Y = MAX(1,NUMROC(M_Y, MB_Y, MYROW, RSRC_Y, NPROW))
```

In this example, LLD\_A = 2 on P<sub>00</sub> and P<sub>01</sub>, LLD\_A = 1 on P<sub>10</sub> and P<sub>11</sub>,  
LLD\_X = LLD\_Y = 2 on P<sub>00</sub>, and LLD\_X = LLD\_Y = 1 on P<sub>10</sub>.

Global complex Hermitian matrix *A* of order 3 with block size 2 × 2:

$$\begin{array}{c} \text{B,D} \end{array} \begin{array}{ccc} & 0 & 1 \\ 0 & \left[ \begin{array}{c|c} \begin{pmatrix} (1.0, 0.0) & . \\ (3.0, -5.0) & (7.0, 0.0) \end{pmatrix} & \begin{pmatrix} . \\ . \end{pmatrix} \\ \hline \begin{pmatrix} (2.0, 3.0) & (4.0, 8.0) \end{pmatrix} & (6.0, 0.0) \end{array} \right] \\ 1 & \end{array}$$

The following is the 2 × 2 process grid:

B,D	0	1
0	P <sub>00</sub>	P <sub>01</sub>
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for  $A$ :

p,q	0	1
0	( 1.0, . ) ( 3.0,-5.0) ( 7.0, . )	:
1	( 2.0, 3.0) ( 4.0, 8.0)	( 6.0, . )

Global vector  $x$  of size  $3 \times 1$  with block size 2:

B,D	0
0	( 1.0, 2.0) ( 4.0, 0.0)
1	( 3.0, 4.0)

The following is the  $2 \times 1$  process grid:

B,D	0	--
0	P <sub>00</sub>	P <sub>01</sub>
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for  $x$ :

p,q	0
0	( 1.0, 2.0) ( 4.0, 0.0)
1	( 3.0, 4.0)

Global vector  $y$  of size  $3 \times 1$  with block size 2:

B,D	0
0	( 1.0, 0.0) ( 2.0,-1.0)
1	( 2.0, 1.0)

The following is the  $2 \times 1$  process grid:

B,D	0	--
0	P <sub>00</sub>	P <sub>01</sub>
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for  $y$ :

p,q	0
0	( 1.0, 0.0) ( 2.0,-1.0)
1	( 2.0, 1.0)

Output:

Global complex Hermitian matrix *A* of order 3 with block size 2 × 2:

B,D	0	1
0	$\begin{pmatrix} (3.0, 0.0) & . \\ (7.0, -10.0) & (23.0, 0.0) \end{pmatrix}$	$\begin{pmatrix} . \\ . \end{pmatrix}$
1	$\begin{pmatrix} (9.0, 4.0) & (14.0, 23.0) \end{pmatrix}$	$(26.0, 0.0)$

The following is the 2 × 2 process grid:

B,D	0	1
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$

Local arrays for *A*:

p,q	0	1
0	$\begin{pmatrix} (3.0, 0.0) & . \\ (7.0, -10.0) & (23.0, 0.0) \end{pmatrix}$	$\begin{pmatrix} . \\ . \end{pmatrix}$
1	$(9.0, 4.0) \ (14.0, 23.0)$	$(26.0, 0.0)$

## PDTRMV and PZTRMV — Matrix-Vector Product for a Triangular Matrix or Its Transpose

### Purpose

PDTRMV computes one of the following matrix-vector products:

- 1.  $x \leftarrow Ax$
- 2.  $x \leftarrow A^T x$

PZTRMV computes one of the following matrix-vector products:

- 1.  $x \leftarrow Ax$
- 2.  $x \leftarrow A^T x$
- 3.  $x \leftarrow A^H x$

where, in the formulas above:

- $A$  represents the global triangular submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$ .
- $x$  represents the global vector:
  - For  $incx = M\_X$ , it is  $X_{ix:ix, jx:jx+n-1}$ .
  - For  $incx = 1$  and  $incx \neq M\_X$ , it is  $X_{ix:ix+n-1, jx:jx}$ .

**Note:** No data should be moved to form  $A^T$  or  $A^H$ ; that is, the matrix  $A$  should always be stored in its untransposed form.

If  $n = 0$ , no computation is performed and the subroutine returns after doing some parameter checking. See references [15] and [16].

Table 54. Data Types

$A, x$	Subprogram
Long-precision real	PDTRMV
Long-precision complex	PZTRMV

### Syntax

<b>Fortran</b>	CALL PDTRMV   PZTRMV ( <i>uplo, transa, diag, n, a, ia, ja, desc_a, x, ix, jx, desc_x, incx</i> )
<b>C and C++</b>	pdtrmv   pztrmv ( <i>uplo, transa, diag, n, a, ia, ja, desc_a, x, ix, jx, desc_x, incx</i> );

### On Entry

*uplo* indicates whether the upper or lower triangular part of the global triangular submatrix  $A$  is referenced, where:

If *uplo* = 'U', the upper triangular part is referenced.

If *uplo* = 'L', the lower triangular part is referenced.

Scope: **global**

Specified as: a single character; *uplo* = 'U' or 'L'.

*transa* indicates the form of matrix  $A$  to use in the computation, where:

If *transa* = 'N',  $A$  is used in the computation.

If *transa* = 'T',  $A^T$  is used in the computation.

If *transa* = 'C',  $A^H$  is used in the computation.

Scope: **global**

Specified as: a single character; *transa* = 'N', 'T', or 'C'.

*diag* indicates the characteristics of the diagonal of matrix *A*, where:

If *diag* = 'U', *A* is a unit triangular matrix.

If *diag* = 'N', *A* is not a unit triangular matrix.

Scope: **global**

Specified as: a single character; *diag* = 'U' or 'N'.

*n* is the order of global triangular submatrix *A* and the length of global vector *x*.

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

*a* is the local part of the global triangular matrix *A*. This identifies the **first element** of the local array *A*. This subroutine computes the location of the first element of the local subarray used, based on *ia*, *ja*, *desc\_a*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading LOCp(*ia*+*n*-1) by LOCq(*ja*+*n*-1) part of the local array *A* must contain the local pieces of the leading *ia*+*n*-1 by *ja*+*n*-1 part of the global matrix, and:

- If *uplo* = 'U', the leading  $n \times n$  upper triangular part of the global triangular submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  must contain the upper triangular part of the submatrix, and the strictly lower triangular part is not referenced.
- If *uplo* = 'L', the leading lower triangular part of the global triangular submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  must contain the lower triangular part of the submatrix, and the strictly upper triangular part is not referenced.

**Note:** No data should be moved to form  $A^T$  or  $A^H$ ; that is, the matrix *A* should always be stored in its untransposed form.

Scope: **local**

Specified as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 54 on page 212. Details about the block-cyclic data distribution of global matrix *A* are stored in *desc\_a*.

*ia* is the row index of the global matrix *A*, identifying the first row of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia$  and  $ia+n-1 \leq M\_A$ .

*ja* is the column index of the global matrix *A*, identifying the first column of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja$  and  $ja+n-1 \leq N\_A$ .

*desc\_a* is the array descriptor for global matrix *A*, described in the following table:

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global

<i>desc_a</i>	Name	Description	Limits	Scope
3	M_A	Number of rows in the global matrix	If $n = 0$ : M_A $\geq 0$ Otherwise: M_A $\geq 1$	Global
4	N_A	Number of columns in the global matrix	If $n = 0$ : N_A $\geq 0$ Otherwise: M_A $\geq 1$	Global
5	MB_A	Row block size	MB_A $\geq 1$	Global
6	NB_A	Column block size	NB_A $\geq 1$	Global
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq \text{RSRC\_A} < p$	Global
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq \text{CSRC\_A} < q$	Global
9	LLD_A	The leading dimension of the local array	LLD_A $\geq \max(1, \text{LOCp}(\text{M\_A}))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

$x$  is the local part of the global matrix  $X$ . This identifies the **first element** of the local array  $X$ . This subroutine computes the location of the first element of the local subarray used, based on  $ix$ ,  $jx$ ,  $desc\_x$ ,  $p$ ,  $q$ ,  $myrow$ , and  $mycol$ ; therefore:

- If  $incx = \text{M\_X}$ , the leading  $\text{LOCp}(ix)$  by  $\text{LOCq}(jx+n-1)$  part of the local array  $X$  must contain the local pieces of the leading  $ix$  by  $jx+n-1$  part of the global matrix.
- If  $incx = 1$  and  $incx \neq \text{M\_X}$ , the leading  $\text{LOCp}(ix+n-1)$  by  $\text{LOCq}(jx)$  part of the local array  $X$  must contain the local pieces of the leading  $ix+n-1$  by  $jx$  part of the global matrix.

Scope: **local**

Specified as: an LLD\_X by (at least)  $\text{LOCq}(\text{N\_X})$  array, containing numbers of the data type indicated in Table 54 on page 212. Details about the block-cyclic data distribution of the global matrix  $X$  are stored in  $desc\_x$ .

$ix$  has the following meaning:

If  $incx = \text{M\_X}$ , it indicates which row of global matrix  $X$  is used for vector  $x$ .

If  $incx = 1$  and  $incx \neq \text{M\_X}$ , it is the row index of global matrix  $X$ , identifying the first element of vector  $x$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ix \leq \text{M\_X}$  and:

If  $incx = 1$  and  $incx \neq \text{M\_X}$ , then  $ix+n-1 \leq \text{M\_X}$ .

$jx$  has the following meaning:

If  $incx = \text{M\_X}$ , it is the column index of global matrix  $X$ , identifying the first element of vector  $x$ .



If  $incx = 1$  and  $incx \neq M\_X$ , it indicates which column of global matrix  $X$  is used for vector  $x$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq jx \leq N\_X$  and:

If  $incx = M\_X$ , then  $jx+n-1 \leq N\_X$ .

$desc\_x$  is the array descriptor for global matrix  $X$ , described in the following table:

$desc\_x$	Name	Description	Limits	Scope
1	DTYPE_X	Descriptor type	DTYPE_X=1	Global
2	CTXT_X	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_X	Number of rows in the global matrix	If $n = 0$ : $M\_X \geq 0$ Otherwise: $M\_X \geq 1$	Global
4	N_X	Number of columns in the global matrix	If $n = 0$ : $N\_X \geq 0$ Otherwise: $M\_X \geq 1$	Global
5	MB_X	Row block size	$MB\_X \geq 1$	Global
6	NB_X	Column block size	$NB\_X \geq 1$	Global
7	RSRC_X	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_X < p$	Global
8	CSRC_X	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_X < q$	Global
9	LLD_X	The leading dimension of the local array	$LLD\_X \geq \max(1, LOCp(M\_X))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

$incx$  is the stride for global vector  $x$ .

Scope: **global**

Specified as: a fullword integer;  $incx = 1$  or  $incx = M\_X$ , where:

If  $incx = M\_X$ , then  $x$  is a row-distributed vector.

If  $incx = 1$  and  $incx \neq M\_X$ , then  $x$  is a column-distributed vector.

## On Return

$x$  is the updated local part of the global matrix  $X$ , containing the results of the computation.

Scope: **local**

Returned as: an LLD\_X by (at least) LOCq(N\_X) array, containing numbers of the data type indicated in Table 54 on page 212.

## Notes and Coding Rules

1. These subroutines accept lowercase letters for the *uplo*, *transa*, and *diag* arguments.
2. For PDTRMV, if you specify 'C' for *transa*, it is interpreted as though you specified 'T'.
3. The matrix and vector must have no common elements; otherwise, results are unpredictable.
4. PDTRMV and PZTRMV assume certain values in your array for parts of a triangular matrix. For unit triangular matrices, the elements of the diagonal are assumed to be one. When using an upper or lower triangular matrix, the unreferenced elements in the strictly lower or upper triangular part, respectively, are assumed to be zero. As a result, you do not have to set these values.
5. The NUMROC utility subroutine can be used to determine the values of LOCp(M\_) and LOCq(N\_) used in the argument descriptions above. For details, see "Determining the Number of Rows and Columns in Your Local Arrays" on page 28 and "NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process" on page 868.
6. For suggested block sizes, see "Coding Tips for Optimizing Parallel Performance" on page 77.
7. The following values must be equal: CTXT\_A = CTXT\_X.
8. The global triangular matrix *A* must be distributed using a square block-cyclic distribution; that is, MB\_A = NB\_A.
9. The block row and block column offsets of the global triangular matrix *A* must be equal; that is,  $\text{mod}(ia-1, MB\_A) = \text{mod}(ja-1, NB\_A)$ .
10. If *incx* = M\_X:
  - The following block sizes must be equal: NB\_X = MB\_A = NB\_A
  - In the process grid, the process column containing the first column of the submatrix *A* must also contain the first column of the submatrix *X*; that is, *iacol* = *ixcol*, where:
    - $iacol = \text{mod}(((ja-1)/NB\_A)+CSRC\_A), q)$
    - $ixcol = \text{mod}(((jx-1)/NB\_X)+CSRC\_X), q)$
  - The block column offset of *x* must be equal to the block row and block column offsets of *A*; that is,  $\text{mod}(jx-1, NB\_X) = \text{mod}(ja-1, NB\_A) = \text{mod}(ia-1, MB\_A)$ .
11. If *incx* = 1( ≠ M\_X):
  - The following block sizes must be equal: MB\_X = MB\_A = NB\_A
  - In the process grid, the process row containing the first row of the submatrix *A* must also contain the first row of the submatrix *X*; that is, *iarow* = *ixrow*, where:
    - $iarow = \text{mod}(((ia-1)/MB\_A)+RSRC\_A), p)$
    - $ixrow = \text{mod}(((ix-1)/MB\_X)+RSRC\_X), p)$
  - The block row offset of *x* must be equal to the block row and block column offsets of *A*; that is,  $\text{mod}(ix-1, MB\_X) = \text{mod}(ia-1, MB\_A) = \text{mod}(ja-1, NB\_A)$ .

## Error Conditions

### Computational Errors

None

**Resource Errors**

Unable to allocate work space

**Input-Argument and Miscellaneous Errors****Stage 1:**

1. DTYPE\_A is invalid.
2. DTYPE\_X is invalid.

**Stage 2:**

1. CTXT\_A is invalid.

**Stage 3:**

1. This subroutine was called from outside the process grid.

**Stage 4:**

1. *uplo*  $\neq$  'U' or 'L'
2. *transa*  $\neq$  'N', 'T', or 'C'
3. *diag*  $\neq$  'N' or 'U'
4.  $n < 0$
5.  $M\_A < 0$  and  $n = 0$ ;  $M\_A < 1$  otherwise
6.  $N\_A < 0$  and  $n = 0$ ;  $N\_A < 1$  otherwise
7.  $MB\_A < 1$
8.  $NB\_A < 1$
9.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
10.  $CSRC\_A < 0$  or  $CSRC\_A \geq q$
11.  $CTXT\_A \neq CTXT\_X$
12.  $M\_X < 0$  and  $n = 0$ ;  $M\_X < 1$  otherwise
13.  $N\_X < 0$  and  $n = 0$ ;  $N\_X < 1$  otherwise
14.  $MB\_X < 1$
15.  $NB\_X < 1$
16.  $RSRC\_X < 0$  or  $RSRC\_X \geq p$
17.  $CSRC\_X < 0$  or  $CSRC\_X \geq q$

**Stage 5:**

1.  $MB\_A = NB\_A$
2.  $\text{mod}(ia-1, MB\_A) \neq \text{mod}(ja-1, NB\_A)$   
If  $n \neq 0$ :
3.  $ix > M\_X$
4.  $jx > N\_X$
5.  $ia > M\_A$
6.  $ja > N\_A$
7.  $ia+n-1 > M\_A$
8.  $ja+n-1 > N\_A$   
If  $incx = M\_X$ :
9.  $NB\_A \neq NB\_X$
10.  $\text{mod}(jx-1, NB\_X) \neq \text{mod}(ja-1, NB\_A)$
11.  $n \neq 0$  and  $jx+n-1 > N\_X$   
If  $incx = 1(\neq M\_X)$ :
12.  $MB\_A \neq MB\_X$
13.  $\text{mod}(ix-1, MB\_X) \neq \text{mod}(ia-1, MB\_A)$
14.  $n \neq 0$  and  $ix+n-1 > M\_X$   
Otherwise:
15.  $incx \neq 1$  and  $incx \neq M\_X$

**Stage 6:**

1.  $LLD\_A < \max(1, LOCp(M\_A))$
2.  $LLD\_X < \max(1, LOCp(M\_X))$
3. If  $incx = M\_X$ , then (in the process grid) the process column containing the first column of the submatrix  $A$  does not contain the first column of the submatrix  $X$ ; that is,  $iacol \neq ixcol$ , where:
  - $iacol = \text{mod}(((ja-1)/NB\_A)+CSRC\_A), q)$
  - $ixcol = \text{mod}(((jx-1)/NB\_X)+CSRC\_X), q)$
4. If  $incx = 1 (\neq M\_X)$ , then (in the process grid) the process row containing the first row of the submatrix  $A$  does not contain the first row of the submatrix  $X$ ; that is,  $iarow \neq ixrow$ , where:
  - $iarow = \text{mod}(((ia-1)/MB\_A)+RSRC\_A), p)$
  - $ixrow = \text{mod}(((ix-1)/MB\_X)+RSRC\_X), p)$

## Examples

### Example 1

This example computes  $x = Ax$  using a  $2 \times 2$  process grid. It uses a global submatrix  $A$  within a global matrix  $A$  by specifying  $ia = 2$  and  $ja = 2$ . It uses vector  $x$ , which is a column-distributed vector within a column of  $X$ , by specifying  $incx = 1$ ,  $ix = 2$ , and  $jx = 1$ .

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      UPLO  TRANSA  DIAG  N    A    IA  JA  DESC_A  X  IX  JX
      |    |      |    |    |    |  |    |    |    |
CALL PDTRMV( 'U' , 'N' , 'N' , 12 , A , 2 , 2 , DESC_A , X , 2 , 1 ,

      DESC_X  INCX
      |      |
      DESC_X , 1 )
```

	Desc_A	Desc_X
DTYPE_	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	13	13
N_	13	1
MB_	3	3
NB_	3	1
RSRC_	0	0
CSRC_	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>

#### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:
 

```
LLD_A = MAX(1,NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_X = MAX(1,NUMROC(M_X, MB_X, MYROW, RSRC_X, NPROW))
```

In this example,  $LLD\_A = 7$  on  $P_{00}$  and  $P_{01}$ ,  $LLD\_A = 6$  on  $P_{10}$  and  $P_{11}$ ,  $LLD\_X = 7$  on  $P_{00}$ , and  $LLD\_X = 6$  on  $P_{10}$ .

After the global matrix  $A$  is distributed over the process grid, only a portion of the global data structure is used—that is, global submatrix  $A$ . Following is the global submatrix  $A$  of order 12, starting at row 2 and column 2 in global triangular matrix  $A$  of order 13 with block size  $3 \times 3$ :

B,D	0	1	2	3	4
0	$\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & 1.0 & 2.0 \\ \cdot & \cdot & 3.0 \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot & \cdot \\ 1.0 & 2.0 & 1.0 \\ 2.0 & 3.0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot & \cdot \\ 1.0 & 3.0 & 1.0 \\ 2.0 & 3.0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot & \cdot \\ 1.0 & 2.0 & 3.0 \\ 1.0 & 2.0 & 3.0 \end{bmatrix}$	$\begin{bmatrix} \cdot \\ 2.0 \\ 3.0 \end{bmatrix}$
1	$\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$	$\begin{bmatrix} 3.0 & 1.0 & 3.0 \\ \cdot & 1.0 & 2.0 \\ \cdot & \cdot & 2.0 \end{bmatrix}$	$\begin{bmatrix} 2.0 & 1.0 & 2.0 \\ 2.0 & 1.0 & 1.0 \\ 1.0 & 2.0 & 2.0 \end{bmatrix}$	$\begin{bmatrix} 1.0 & 2.0 & 3.0 \\ 1.0 & 2.0 & 3.0 \\ 1.0 & 2.0 & 3.0 \end{bmatrix}$	$\begin{bmatrix} 1.0 \\ 2.0 \\ 3.0 \end{bmatrix}$
2	$\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$	$\begin{bmatrix} 1.0 & 2.0 & 1.0 \\ \cdot & 2.0 & 1.0 \\ \cdot & \cdot & 2.0 \end{bmatrix}$	$\begin{bmatrix} 1.0 & 2.0 & 3.0 \\ 1.0 & 2.0 & 3.0 \\ 1.0 & 2.0 & 3.0 \end{bmatrix}$	$\begin{bmatrix} 1.0 \\ 2.0 \\ 3.0 \end{bmatrix}$
3	$\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$	$\begin{bmatrix} 3.0 & 1.0 & 3.0 \\ \cdot & 2.0 & 2.0 \\ \cdot & \cdot & 1.0 \end{bmatrix}$	$\begin{bmatrix} 1.0 \\ 2.0 \\ 3.0 \end{bmatrix}$
4	$\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$	$\begin{bmatrix} 1.0 \\ 1.0 \\ 1.0 \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2 4	1 3
0	$P_{00}$	$P_{01}$
2		
4		
1	$P_{10}$	$P_{11}$
3		

Local arrays for  $A$ :

p,q	0	1
0	$\begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & 1.0 & 2.0 & 1.0 & 3.0 & 1.0 & 2.0 \\ \cdot & \cdot & 3.0 & 2.0 & 3.0 & 1.0 & 3.0 \\ \cdot & \cdot & \cdot & 1.0 & 2.0 & 1.0 & 1.0 \\ \cdot & \cdot & \cdot & \cdot & 2.0 & 1.0 & 2.0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & 2.0 & 3.0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1.0 \end{bmatrix}$	$\begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 1.0 & 2.0 & 1.0 & 1.0 & 2.0 & 3.0 \\ 2.0 & 3.0 & 1.0 & 1.0 & 2.0 & 3.0 \\ \cdot & \cdot & \cdot & 1.0 & 2.0 & 3.0 \\ \cdot & \cdot & \cdot & 1.0 & 2.0 & 3.0 \\ \cdot & \cdot & \cdot & 1.0 & 2.0 & 3.0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}$
1	$\begin{bmatrix} \cdot & \cdot & \cdot & 2.0 & 1.0 & 2.0 & 1.0 \\ \cdot & \cdot & \cdot & 2.0 & 1.0 & 1.0 & 2.0 \\ \cdot & \cdot & \cdot & 1.0 & 2.0 & 2.0 & 3.0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1.0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 2.0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 3.0 \end{bmatrix}$	$\begin{bmatrix} 3.0 & 1.0 & 3.0 & 1.0 & 2.0 & 3.0 \\ \cdot & 1.0 & 2.0 & 1.0 & 2.0 & 3.0 \\ \cdot & \cdot & 2.0 & 1.0 & 2.0 & 3.0 \\ \cdot & \cdot & \cdot & 3.0 & 1.0 & 3.0 \\ \cdot & \cdot & \cdot & \cdot & 2.0 & 2.0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1.0 \end{bmatrix}$

After the global matrix  $X$  is distributed over the process grid, only a portion of the global data structure is used—that is, global vector  $x$ , which is a column-distributed vector. Following is the global vector  $x$  of size  $12 \times 1$ , starting at row 2 in  $13 \times 1$  global matrix  $X$  with block size  $3 \times 1$ :

## PDTRMV and PZTRMV

B,D	0
	$\begin{bmatrix} . \\ 2.0 \\ 3.0 \\ \hline 1.0 \\ 2.0 \\ 3.0 \\ \hline 1.0 \\ 2.0 \\ 3.0 \\ \hline 1.0 \\ 2.0 \\ 3.0 \\ \hline 1.0 \end{bmatrix}$
0	
1	
2	
3	
4	

The following is the  $2 \times 2$  process grid:

B,D	0	--
0	$P_{00}$	$P_{01}$
2		
4		
1	$P_{10}$	$P_{11}$
3		

Local arrays for  $x$ :

p,q	0
	$\begin{bmatrix} . \\ 2.0 \\ 3.0 \\ \hline 1.0 \\ 2.0 \\ 3.0 \\ \hline 1.0 \end{bmatrix}$
0	
1	

### Output:

After the global matrix  $X$  is distributed over the process grid, only a portion of the global data structure is used—that is, global vector  $x$ , which is a column-distributed vector. Following is the global vector  $x$  of size  $12 \times 1$ , starting at row 2 in  $13 \times 1$  global matrix  $X$  with block size  $3 \times 1$ :

B,D	0
	$\begin{bmatrix} . \\ 42.0 \\ 48.0 \\ \hline 39.0 \\ 31.0 \\ 34.0 \end{bmatrix}$
0	
1	

2	----
	23.0
	23.0
3	----
	15.0
	12.0
4	----
	6.0
	1.0

The following is the  $2 \times 2$  process grid:

B,D	0	--
0	P <sub>00</sub>	P <sub>01</sub>
2		
4		
1	P <sub>10</sub>	P <sub>11</sub>
3		

Local arrays for  $x$ :

p,q	0
	----
0	.
	42.0
	48.0
	23.0
	23.0
	23.0
1	1.0
	39.0
	31.0
	34.0
	15.0
	12.0
	6.0

## Example 2

This example computes  $x = Ax$  using a  $2 \times 2$  process grid.

**Note:** For unit triangular matrices, the elements of the diagonal are assumed to be one, so you do not have to set these values.

### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      UPLO  TRANSA  DIAG  N    A    IA  JA  DESC_A  X  IX  JX
CALL PZTRMV( 'L' , 'N' , 'U' , 4 , A , 1 , 1 , DESC_A , X , 1 , 1 ,
              DESC_X INCX
              DESC_X , 1 )
```

	Desc_A	Desc_X
DTYPE_	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	4	4
N_	4	1
MB_	2	2
NB_	2	1
RSRC_	0	0
CSRC_	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:  

$$\text{LLD\_A} = \text{MAX}(1, \text{NUMROC}(\text{M\_A}, \text{MB\_A}, \text{MYROW}, \text{RSRC\_A}, \text{NPROW}))$$

$$\text{LLD\_X} = \text{MAX}(1, \text{NUMROC}(\text{M\_X}, \text{MB\_X}, \text{MYROW}, \text{RSRC\_X}, \text{NPROW}))$$

In this example, LLD\_A = 2 on P<sub>00</sub> and P<sub>01</sub>, LLD\_A = 2 on P<sub>10</sub> and P<sub>11</sub>,  
LLD\_X = 2 on P<sub>00</sub> and LLD\_X = 2 on P<sub>10</sub>.

Global triangular matrix *A* of order 4 with block size 2 × 2:

$$\begin{array}{c} \text{B,D} \end{array} \quad \begin{array}{cc} 0 & 1 \end{array}$$

$$\begin{array}{c} 0 \\ 1 \end{array} \left[ \begin{array}{cc|cc} (1.0,0.0) & . & . & . \\ (1.0,1.0) & (1.0,0.0) & . & . \\ \hline (1.0,1.0) & (3.0,3.0) & (1.0,0.0) & . \\ (3.0,3.0) & (4.0,4.0) & (3.0,3.0) & (1.0,0.0) \end{array} \right]$$

The following is the 2 × 2 process grid:

$$\begin{array}{c} \text{B,D} \end{array} \quad \begin{array}{c|c} 0 & 1 \\ \hline 0 & P_{00} \quad P_{01} \\ \hline 1 & P_{10} \quad P_{11} \end{array}$$

Local arrays for *A*:

$$\begin{array}{c} \text{p,q} \end{array} \quad \begin{array}{cc|cc} 0 & 1 \\ \hline 0 & . & . & . \\ (1.0,1.0) & . & . & . \\ \hline 1 & (1.0,1.0) & (3.0,3.0) & . \\ (3.0,3.0) & (4.0,4.0) & (3.0,3.0) & . \end{array}$$

Global vector *x* of size 4 × 1 with block size 2:

$$\begin{array}{c} \text{B,D} \end{array} \quad \begin{array}{c} 0 \end{array}$$

$$\begin{array}{c} 0 \\ 1 \end{array} \left[ \begin{array}{c} (1.0,1.0) \\ (2.0,2.0) \\ \hline (3.0,3.0) \\ (4.0,4.0) \end{array} \right]$$



The following is the  $2 \times 2$  process grid:

B,D	0	--
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$

Local arrays for  $x$ :

p,q	0
0	(1.0, 1.0) (2.0, 2.0)
1	(3.0, 3.0) (4.0, 4.0)

**Output:**

Global vector  $x$  of size  $4 \times 1$  with block size 2:

B,D	0
0	( 1.0, 1.0) ( 2.0, 4.0)
1	( 3.0, 17.0) ( 4.0, 44.0)

The following is the  $2 \times 2$  process grid:

B,D	0	--
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$

Local arrays for  $x$ :

p,q	0
0	( 1.0, 1.0) ( 2.0, 4.0)
1	( 3.0, 17.0) ( 4.0, 44.0)

## PDTRSV and PZTRSV — Solution of Triangular System of Equations with a Single Right-Hand Side

### Purpose

PDTRSV performs one of the following solves for a triangular system of equations with a single right-hand side:

Solution	Equation
1. $x \leftarrow A^{-1}x$	$Ax = b$
2. $x \leftarrow A^{-T}x$	$A^T x = b$

PZTRSV performs one of the following solves for a triangular system of equations with a single right-hand side:

Solution	Equation
1. $x \leftarrow A^{-1}x$	$Ax = b$
2. $x \leftarrow A^{-T}x$	$A^T x = b$
3. $x \leftarrow A^{-H}x$	$A^H x = b$

where, in the formulas above:

- $A$  represents the global triangular submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$ .
- $x$  represents the global vector:
  - For  $incx = M\_X$ , it is  $X_{ix:ix, jx:jx+n-1}$ .
  - For  $incx = 1$  and  $incx \neq M\_X$ , it is  $X_{ix:ix+n-1, jx:jx}$ .

#### Notes:

1. The term  $b$  used in the systems of equations listed above represents the right-hand side of the system. It is important to note that in these subroutines the right-hand side of the equation is actually provided in the input-output argument  $x$ .
2. No data should be moved to form  $A^T$  or  $A^H$ ; that is, the matrix  $A$  should always be stored in its untransposed form.

If  $n = 0$ , no computation is performed and the subroutine returns after doing some parameter checking. See references [15] and [16].

Table 55. Data Types

$A, x$	Subprogram
Long-precision real	PDTRSV
Long-precision complex	PZTRSV

### Syntax

Fortran	CALL PDTRSV   PZTRSV ( <i>uplo</i> , <i>transa</i> , <i>diag</i> , <i>n</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>x</i> , <i>ix</i> , <i>jx</i> , <i>desc_x</i> , <i>incx</i> )
C and C++	pdtrsv   pztrsv ( <i>uplo</i> , <i>transa</i> , <i>diag</i> , <i>n</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>x</i> , <i>ix</i> , <i>jx</i> , <i>desc_x</i> , <i>incx</i> );

### On Entry

*uplo* indicates whether the upper or lower triangular part of the global triangular submatrix  $A$  is referenced, where:

If *uplo* = 'U', the upper triangular part is referenced.

If  $uplo = 'L'$ , the lower triangular part is referenced.

Scope: **global**

Specified as: a single character;  $uplo = 'U'$  or  $'L'$ .

*transa* indicates the form of matrix  $A$  used in the system of equations, where:

If  $transa = 'N'$ ,  $A$  is used in the system of equations.

If  $transa = 'T'$ ,  $A^T$  is used in the system of equations.

If  $transa = 'C'$ ,  $A^H$  is used in the system of equations.

Scope: **global**

Specified as: a single character;  $transa = 'N'$ ,  $'T'$ , or  $'C'$ .

*diag* indicates the characteristics of the diagonal of matrix  $A$ , where:

If  $diag = 'U'$ ,  $A$  is a unit triangular matrix.

If  $diag = 'N'$ ,  $A$  is not a unit triangular matrix.

Scope: **global**

Specified as: a single character;  $diag = 'U'$  or  $'N'$ .

*n* is the order of global triangular submatrix  $A$  and the length of global vector  $x$ .

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

*a* is the local part of the global triangular matrix  $A$ , used in the system of equations. This identifies the **first element** of the local array  $A$ . This subroutine computes the location of the first element of the local subarray used, based on *ia*, *ja*, *desc\_a*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading  $LOCp(ia+n-1)$  by  $LOCq(ja+n-1)$  part of the local array  $A$  must contain the local pieces of the leading  $ia+n-1$  by  $ja+n-1$  part of the global matrix, and:

- If  $uplo = 'U'$ , the leading  $n \times n$  upper triangular part of the global triangular submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  must contain the upper triangular part of the submatrix, and the strictly lower triangular part is not referenced.
- If  $uplo = 'L'$ , the leading lower triangular part of the global triangular submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  must contain the lower triangular part of the submatrix, and the strictly upper triangular part is not referenced.

**Note:** No data should be moved to form  $A^T$  or  $A^H$ ; that is, the matrix  $A$  should always be stored in its untransposed form.

Scope: **local**

Specified as: an  $LLD\_A$  by (at least)  $LOCq(N\_A)$  array, containing numbers of the data type indicated in Table 55 on page 224. Details about the block-cyclic data distribution of global matrix  $A$  are stored in *desc\_a*.

*ia* is the row index of the global matrix  $A$ , identifying the first row of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia$  and  $ia+n-1 \leq M\_A$ .

*ja* is the column index of the global matrix  $A$ , identifying the first column of the submatrix  $A$ .

## PDTRSV and PZTRSV

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja$  and  $ja+n-1 \leq N\_A$ .

*desc\_a* is the array descriptor for global matrix *A*, described in the following table:

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_A < p$	Global
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < q$	Global
9	LLD_A	The leading dimension of the local array	$LLD\_A \geq \max(1, LOCp(M\_A))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

*x* is the local part of the global matrix *X*, containing the right-hand side of the triangular system to be solved. This identifies the **first element** of the local array *x*. This subroutine computes the location of the first element of the local subarray used, based on *ix*, *jx*, *desc\_x*, *p*, *q*, *myrow*, and *mycol*; therefore:

- If  $incx = M\_X$ , the leading  $LOCp(ix)$  by  $LOCq(jx+n-1)$  part of the local array *x* must contain the local pieces of the leading *ix* by  $jx+n-1$  part of the global matrix.
- If  $incx = 1$  and  $incx \neq M\_X$ , the leading  $LOCp(ix+n-1)$  by  $LOCq(jx)$  part of the local array *x* must contain the local pieces of the leading  $ix+n-1$  by *jx* part of the global matrix.

Scope: **local**

Specified as: an  $LLD\_X$  by (at least)  $LOCq(N\_X)$  array, containing numbers of the data type indicated in Table 55 on page 224. Details about the block-cyclic data distribution of the global matrix *X* are stored in *desc\_x*.

*ix* has the following meaning:

If  $incx = M\_X$ , it indicates which row of global matrix *X* is used for vector *x*.

If  $incx = 1$  and  $incx \neq M\_X$ , it is the row index of global matrix  $X$ , identifying the first element of vector  $x$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ix \leq M\_X$  and:

If  $incx = 1$  and  $incx \neq M\_X$ , then  $ix+n-1 \leq M\_X$ .

$jx$  has the following meaning:

If  $incx = M\_X$ , it is the column index of global matrix  $X$ , identifying the first element of vector  $x$ .

If  $incx = 1$  and  $incx \neq M\_X$ , it indicates which column of global matrix  $X$  is used for vector  $x$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq jx \leq N\_X$  and:

If  $incx = M\_X$ , then  $jx+n-1 \leq N\_X$ .

$desc\_x$  is the array descriptor for global matrix  $X$ , described in the following table:

$desc\_x$	Name	Description	Limits	Scope
1	DTYPE_X	Descriptor type	DTYPE_X=1	Global
2	CTXT_X	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_X	Number of rows in the global matrix	If $n = 0$ : $M\_X \geq 0$ Otherwise: $M\_X \geq 1$	Global
4	N_X	Number of columns in the global matrix	If $n = 0$ : $N\_X \geq 0$ Otherwise: $M\_X \geq 1$	Global
5	MB_X	Row block size	$MB\_X \geq 1$	Global
6	NB_X	Column block size	$NB\_X \geq 1$	Global
7	RSRC_X	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_X < p$	Global
8	CSRC_X	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_X < q$	Global
9	LLD_X	The leading dimension of the local array	$LLD\_X \geq \max(1, LOCP(M\_X))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

$incx$  is the stride for global vector  $x$ .

Scope: **global**

Specified as: a fullword integer;  $incx = 1$  or  $incx = M\_X$ , where:

If  $incx = M\_X$ , then  $x$  is a row-distributed vector.

If  $incx = 1$  and  $incx \neq M\_X$ , then  $x$  is a column-distributed vector.

**On Return**

$x$  is the updated local part of the global matrix  $X$ , containing the solution vector.

Scope: **local**

Returned as: an LLD\_X by (at least) LOCq(N\_X) array, containing numbers of the data type indicated in Table 55 on page 224.

**Notes and Coding Rules**

1. These subroutines accept lowercase letters for the *uplo*, *transa*, and *diag* arguments.
2. For PDTRSV, if you specify 'C' for *transa*, it is interpreted as though you specified 'T'.
3. The matrix and vector must have no common elements; otherwise, results are unpredictable.
4. PDTRSV and PZTRSV assume certain values in your array for parts of a triangular matrix. For unit triangular matrices, the elements of the diagonal are assumed to be one. When using an upper or lower triangular matrix, the unreferenced elements in the strictly lower or upper triangular part, respectively, are assumed to be zero. As a result, you do not have to set these values.
5. The NUMROC utility subroutine can be used to determine the values of LOCp(M\_) and LOCq(N\_) used in the argument descriptions above. For details, see "Determining the Number of Rows and Columns in Your Local Arrays" on page 28 and "NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process" on page 868.
6. For suggested block sizes, see "Coding Tips for Optimizing Parallel Performance" on page 77.
7. The following values must be equal: CTXT\_A = CTXT\_X.
8. The global triangular matrix  $A$  must be distributed using a square block-cyclic distribution; that is, MB\_A = NB\_A.
9. The block row and block column offsets of the global triangular matrix  $A$  must be equal; that is,  $\text{mod}(ia-1, MB_A) = \text{mod}(ja-1, NB_A)$ .
10. If  $incx = M_X$ :
  - The following block sizes must be equal: NB\_X = MB\_A = NB\_A
  - If *transa* = 'T', then (in the process grid) the process column containing the first column of the submatrix  $A$  must also contain the first column of the submatrix  $X$ ; that is,  $iacol = ixcol$ , where:
    - $iacol = \text{mod}(((ja-1)/NB_A)+CSRC_A), q)$
    - $ixcol = \text{mod}(((jx-1)/NB_X)+CSRC_X), q)$
  - The block column offset of  $x$  must be equal to the block row and block column offsets of  $A$ ; that is,  $\text{mod}(jx-1, NB_X) = \text{mod}(ja-1, NB_A) = \text{mod}(ia-1, MB_A)$ .
11. If  $incx = 1$  (≠ M\_X):
  - The following block sizes must be equal: MB\_X = MB\_A = NB\_A
  - If *transa* = 'N', then (in the process grid) the process row containing the first row of the submatrix  $A$  must also contain the first row of the submatrix  $X$ ; that is,  $iarow = ixrow$ , where:
    - $iarow = \text{mod}(((ia-1)/MB_A)+RSRC_A), p)$
    - $ixrow = \text{mod}(((ix-1)/MB_X)+RSRC_X), p)$

- The block row offset of  $x$  must be equal to the block row and block column offsets of  $A$ ; that is,  $\text{mod}(ix-1, MB\_X) = \text{mod}(ia-1, MB\_A) = \text{mod}(ja-1, NB\_A)$ .

## Error Conditions

### Computational Errors

None

### Resource Errors

Unable to allocate work space

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1.  $DTYPE\_A$  is invalid.
2.  $DTYPE\_X$  is invalid.

#### Stage 2:

1.  $CTXT\_A$  is invalid.

#### Stage 3:

1. This subroutine was called from outside the process grid.

#### Stage 4:

1.  $uplo \neq 'U'$  or  $'L'$
2.  $transa \neq 'N', 'T',$  or  $'C'$
3.  $diag \neq 'N'$  or  $'U'$
4.  $n < 0$
5.  $M\_A < 0$  and  $n = 0$ ;  $M\_A < 1$  otherwise
6.  $N\_A < 0$  and  $n = 0$ ;  $N\_A < 1$  otherwise
7.  $MB\_A < 1$
8.  $NB\_A < 1$
9.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
10.  $CSRC\_A < 0$  or  $CSRC\_A \geq q$
11.  $CTXT\_A \neq CTXT\_X$
12.  $M\_X < 0$  and  $n = 0$ ;  $M\_X < 1$  otherwise
13.  $N\_X < 0$  and  $n = 0$ ;  $N\_X < 1$  otherwise
14.  $MB\_X < 1$
15.  $NB\_X < 1$
16.  $RSRC\_X < 0$  or  $RSRC\_X \geq p$
17.  $CSRC\_X < 0$  or  $CSRC\_X \geq q$

#### Stage 5:

1.  $MB\_A \neq NB\_A$
2.  $\text{mod}(ia-1, MB\_A) \neq \text{mod}(ja-1, NB\_A)$   
If  $n \neq 0$ :
3.  $ix > M\_X$
4.  $jx > N\_X$
5.  $ia > M\_A$
6.  $ja > N\_A$
7.  $ia+n-1 > M\_A$
8.  $ja+n-1 > N\_A$   
If  $incx = M\_X$ :
9.  $NB\_A \neq NB\_X$
10.  $\text{mod}(jx-1, NB\_X) \neq \text{mod}(ja-1, NB\_A)$

11.  $n \neq 0$  and  $jx+n-1 > N\_X$   
If  $incx = 1$  ( $\neq M\_X$ ):
12.  $MB\_A \neq MB\_X$
13.  $\text{mod}(ix-1, MB\_X) \neq \text{mod}(ia-1, MB\_A)$
14.  $n \neq 0$  and  $ix+n-1 > M\_X$   
Otherwise:
15.  $incx \neq 1$  and  $incx \neq M\_X$

**Stage 6:**

1.  $LLD\_A < \max(1, LOCp(M\_A))$
2.  $LLD\_X < \max(1, LOCp(M\_X))$
3. If  $incx = M\_X$  and  $transa = 'T'$ , then (in the process grid) the process column containing the first column of the submatrix  $A$  does not contain the first column of the submatrix  $X$ ; that is,  $iacol \neq ixcol$ , where:
  - $iacol = \text{mod}(((ja-1)/NB\_A)+CSRC\_A), q)$
  - $ixcol = \text{mod}(((jx-1)/NB\_X)+CSRC\_X), q)$
4. If  $incx = 1$  ( $\neq M\_X$ ) and  $transa = 'N'$ , then (in the process grid) the process row containing the first row of the submatrix  $A$  does not contain the first row of the submatrix  $X$ ; that is,  $iarow \neq ixrow$ , where:
  - $iarow = \text{mod}((((ia-1)/MB\_A)+RSRC\_A), p)$
  - $ixrow = \text{mod}((((ix-1)/MB\_X)+RSRC\_X), p)$

## Examples

### Example 1

This example solves  $x \leftarrow A^{-1}x$  using a  $2 \times 2$  process grid, where  $A$  is a unit triangular matrix. It uses a global submatrix  $A$  within a global matrix  $A$  by specifying  $ia = 2$  and  $ja = 2$ . It uses vector  $x$ , which is a row-distributed vector within a row of global matrix  $X$ , by specifying  $incx = M\_X = 1$ ,  $ix = 1$ , and  $jx = 2$ .

**Note:** Because matrix  $A$  is unit triangular, the diagonal elements are not referenced. This subroutine assumes a value of 1.0 for the diagonal elements.

**Call Statements and Input:**

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      UPLO  TRANSA DIAG  N   A   IA  JA   DESC_A  X  IX  JX
      |    |    |    |   |   |   |   |    |   |  |  |
CALL PDTRSV( 'L' , 'N' , 'U' , 12 , A , 2 , 2 , DESC_A , X , 1 , 2 ,

      DESC_X INCX
      |    |
      DESC_X , 1 )
```

	Desc_A	Desc_X
DTYPE_	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	13	1
N_	13	13
MB_	3	1



	Desc_A	Desc_X
NB_	3	3
RSRC_	0	0
CSRC_	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.

2. Each process should set the LLD\_ as follows:

LLD\_A = MAX(1, NUMROC(M\_A, MB\_A, MYROW, RSRC\_A, NPROW))

LLD\_X = MAX(1, NUMROC(M\_X, MB\_X, MYROW, RSRC\_X, NPROW))

In this example, LLD\_A = 7 on P<sub>00</sub> and P<sub>01</sub>, LLD\_A = 6 on P<sub>10</sub> and P<sub>11</sub>, and

LLD\_X = 1 on all processes.

After the global matrix *A* is distributed over the process grid, only a portion of the global data structure is used—that is, global submatrix *A*. Following is the global submatrix *A* of order 12, starting at row 2 and column 2 in global triangular matrix *A* of order 13 with block size 3 × 3:

B,D	0	1	2	3	4
0	. . . . 1.0 . . 2.0 1.0	. . . . . . . . .	. . . . . . . . .	. . . . . . . . .	. . . . .
1	. 3.0 2.0 . 1.0 3.0 . 2.0 1.0	1.0 . . 2.0 1.0 . 3.0 2.0 1.0	. . . . . . . . .	. . . . . . . . .	. . . . .
2	. 3.0 2.0 . 1.0 3.0 . 2.0 1.0	1.0 3.0 2.0 2.0 1.0 3.0 3.0 2.0 1.0	1.0 . . 2.0 1.0 . 3.0 2.0 1.0	. . . . . . . . .	. . . . .
3	. 3.0 2.0 . 1.0 3.0 . 2.0 1.0	1.0 3.0 2.0 2.0 1.0 3.0 3.0 2.0 1.0	1.0 3.0 2.0 2.0 1.0 3.0 3.0 2.0 1.0	1.0 . . 2.0 1.0 . 3.0 2.0 1.0	. . . . .
4	. 3.0 2.0	1.0 3.0 2.0	1.0 3.0 2.0	1.0 3.0 2.0	1.0

The following is the 2 × 2 process grid:

B,D	0 2 4	1 3
0 2 4	P <sub>00</sub>	P <sub>01</sub>
1 3	P <sub>10</sub>	P <sub>11</sub>

Local arrays for *A*:

## PDTRSV and PZTRSV

p,q	0							1					
0	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	2.0	.	.	.	.	.	.	.	.	.	.	.
	.	3.0	2.0	.	.	.	.	1.0	3.0	2.0	.	.	.
	.	1.0	3.0	2.0	.	.	.	2.0	1.0	3.0	.	.	.
	.	2.0	1.0	3.0	2.0	.	.	3.0	2.0	1.0	.	.	.
	.	3.0	2.0	1.0	3.0	2.0	.	1.0	3.0	2.0	1.0	3.0	2.0
1	.	3.0	2.0	.	.	.	.	.	.	.	.	.	.
	.	1.0	3.0	.	.	.	.	2.0	.	.	.	.	.
	.	2.0	1.0	.	.	.	.	3.0	2.0	.	.	.	.
	.	3.0	2.0	1.0	3.0	2.0	.	1.0	3.0	2.0	.	.	.
	.	1.0	3.0	2.0	1.0	3.0	.	2.0	1.0	3.0	2.0	.	.
	.	2.0	1.0	3.0	2.0	1.0	.	3.0	2.0	1.0	3.0	2.0	.
	.	3.0	2.0	1.0	3.0	2.0	.	1.0	3.0	2.0	1.0	3.0	2.0

After the global matrix  $X$  is distributed over the process grid, only a portion of the global data structure is used—that is, global vector  $x$ , which is a row-distributed vector. Following is the global vector  $x$  of size  $1 \times 12$ , starting at row 1 and column 2 in  $1 \times 13$  global matrix  $X$  with block size  $1 \times 3$ :

B,D	0	1	2	3	4
0	$\left[ \begin{array}{ccccc} . & 2.0 & 7.0 &   & 13.0 & 15.0 & 17.0 &   & 26.0 & 28.0 & 27.0 &   & 39.0 & 41.0 & 37.0 &   & 52.0 \end{array} \right]$				

The following is the  $2 \times 2$  process grid:

B,D	0 2 4	1 3
0	$P_{00}$	$P_{01}$
--	$P_{10}$	$P_{11}$

Local arrays for  $x$ :

p,q	0	1
0	. 2.0 7.0 26.0 28.0 27.0 52.0	13.0 15.0 17.0 39.0 41.0 37.0

**Output:**

After the global matrix  $X$  is distributed over the process grid, only a portion of the global data structure is used—that is, global vector  $x$ , which is a row-distributed vector. Following is the global vector  $x$  of size  $1 \times 12$ , starting at row 1 and column 2 in  $1 \times 13$  global matrix  $X$  with block size  $1 \times 3$ :

B,D	0	1	2	3	4
0	$\left[ \begin{array}{ccccc} . & 2.0 & 3.0 &   & 1.0 & 2.0 & 3.0 &   & 1.0 & 2.0 & 3.0 &   & 1.0 & 2.0 & 3.0 &   & 1.0 \end{array} \right]$				

The following is the  $2 \times 2$  process grid:

B,D	0 2 4	1 3
0	$P_{00}$	$P_{01}$
--	$P_{10}$	$P_{11}$

Local arrays for  $x$ :

p,q	0								1					
0	.	2.0	3.0	1.0	2.0	3.0	1.0		1.0	2.0	3.0	1.0	2.0	3.0

## Example 2

This example solves  $x \leftarrow A^{-1}x$  using a  $2 \times 2$  process grid, where  $A$  is a unit triangular matrix.

**Note:** Because matrix  $A$  is unit triangular, the diagonal elements are not referenced. This subroutine assumes a value of (1.0,0.0) for the diagonal elements.

### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      UPLO  TRANSA  DIAG  N   A   IA  JA  DESC_A  X  IX  JX
      |    |      |    |   |   |   |   |    |  |  |  |
CALL PZTRSV( 'L' , 'N' , 'U' , 4 , A , 1 , 1 , DESC_A , X , 1 , 1 ,
      DESC_X  INCX
      |      |
      DESC_X , 1 )
```

	Desc_A	Desc_X
DTYPE_	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	4	4
N_	4	1
MB_	2	2
NB_	2	1
RSRC_	0	0
CSRC_	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>

### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.

2. Each process should set the LLD\_ as follows:

```
LLD_A = MAX(1, NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_X = MAX(1, NUMROC(M_X, MB_X, MYROW, RSRC_X, NPROW))
```

In this example, LLD\_A = 2 on P<sub>00</sub> and P<sub>01</sub>, LLD\_A = 2 on P<sub>10</sub> and P<sub>11</sub>, and LLD\_X = 2 on P<sub>00</sub> and P<sub>10</sub>.

Global triangular matrix  $A$  of order 4 with block size  $2 \times 2$ :

$$B, D \quad \begin{array}{c|c} 0 & 1 \\ \hline 0 & \left[ \begin{array}{cc|cc} (1.0, 0.0) & . & . & . \\ (1.0, 1.0) & (1.0, 0.0) & . & . \\ \hline . & . & . & . \end{array} \right] \end{array}$$

## PDTRSV and PZTRSV

$$1 \quad \left[ \begin{array}{cc|cc} (1.0,1.0) & (3.0,3.0) & (1.0,0.0) & . \\ (3.0,3.0) & (4.0,4.0) & (3.0,3.0) & (1.0,0.0) \end{array} \right]$$

The following is the  $2 \times 2$  process grid:

B,D	0	1
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$

Local arrays for A:

p,q	0	1
0	$\begin{pmatrix} . & . \\ (1.0,1.0) & . \end{pmatrix}$	$\begin{pmatrix} . & . \\ . & . \end{pmatrix}$
1	$\begin{pmatrix} (1.0,1.0) & (3.0,3.0) \\ (3.0,3.0) & (4.0,4.0) \end{pmatrix}$	$\begin{pmatrix} . & . \\ (3.0,3.0) & . \end{pmatrix}$

Global vector  $x$  of size  $4 \times 1$  with block size 2:

B,D	0
0	$\begin{pmatrix} (1.0, 1.0) \\ (2.0, 4.0) \end{pmatrix}$
1	$\begin{pmatrix} (3.0,17.0) \\ (4.0,44.0) \end{pmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0	--
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$

Local arrays for  $x$ :

p,q	0
0	$\begin{pmatrix} (1.0, 1.0) \\ (2.0, 4.0) \end{pmatrix}$
1	$\begin{pmatrix} (3.0,17.0) \\ (4.0,44.0) \end{pmatrix}$

**Output:**

Global vector  $x$  of size  $4 \times 1$  with block size 2:

B,D	0
0	$\begin{pmatrix} (1.0,1.0) \\ (2.0,2.0) \end{pmatrix}$
1	$\begin{pmatrix} (3.0,3.0) \\ (4.0,4.0) \end{pmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0	--
0	P <sub>00</sub>	P <sub>01</sub>
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for *x*:

p,q	0
0	(1.0,1.0) (2.0,2.0)
1	(3.0,3.0) (4.0,4.0)



---

## Chapter 7. Level 3 PBLAS

This chapter describes the Level 3 PBLAS subroutines.

---

### Overview of the Level 3 PBLAS Subroutines

The Level 3 PBLAS include a subset of the standard set of distributed memory parallel versions of the Level 3 BLAS.

**Note:** These subroutines are designed in accordance with the proposed Level 3 PBLAS standard. (See references [15], [16], and [18].) If these subroutines do not comply with the standard as approved, IBM will consider updating them to do so. If IBM updates these subroutines, the update could require modifications of the calling application program.

*Table 56. List of Level 3 PBLAS*

Descriptive Name	Long-Precision Subprogram	Page
Matrix-Matrix Product for a General Matrix, Its Transpose, or Its Conjugate Transpose	PDGEMM PZGEMM	239
Matrix-Matrix Product Where One Matrix is Real or Complex Symmetric or Complex Hermitian	PDSYMM PZSYMM PZHEMM	256
Triangular Matrix-Matrix Product	PDTRMM PZTRMM	276
Solution of Triangular System of Equations with Multiple Right-Hand Sides	PDTRSM PZTRSM	288
Rank-K Update of a Real or Complex Symmetric or a Complex Hermitian Matrix	PDSYRK PZSYRK PZHERK	301
Rank-2K Update of a Real or Complex Symmetric or a Complex Hermitian Matrix	PDSYR2K PZSYR2K PZHER2K	316
Matrix Transpose for a General Matrix	PDTRAN PZTRANC PZTRANU	336

---

## Level 3 PBLAS Subroutines

This section contains the Level 3 PBLAS subroutine descriptions.



## PDGEMM and PZGEMM — Matrix-Matrix Product for a General Matrix, Its Transpose, or Its Conjugate Transpose

### Purpose

PDGEMM performs any one of the following combined matrix computations:

- $C \leftarrow \alpha AB + \beta C$
- $C \leftarrow \alpha AB^T + \beta C$
- $C \leftarrow \alpha A^T B + \beta C$
- $C \leftarrow \alpha A^T B^T + \beta C$

PZGEMM performs any one of the following combined matrix computations:

- $C \leftarrow \alpha AB + \beta C$
- $C \leftarrow \alpha AB^T + \beta C$
- $C \leftarrow \alpha A^T B + \beta C$
- $C \leftarrow \alpha A^T B^T + \beta C$
- $C \leftarrow \alpha A^H B + \beta C$
- $C \leftarrow \alpha A^H B^T + \beta C$
- $C \leftarrow \alpha AB^H + \beta C$
- $C \leftarrow \alpha A^T B^H + \beta C$
- $C \leftarrow \alpha A^H B^H + \beta C$

where, in the PDGEMM and PZGEMM formulas above:

- $A$  represents the global general submatrix:
  - For  $transa = 'N'$ , it is  $A_{ia:ia+m-1, ja:ja+k-1}$ .
  - For  $transa = 'T'$  or  $'C'$ , it is  $A_{ia:ia+k-1, ja:ja+m-1}$ .
- $B$  represents the global general submatrix:
  - For  $transb = 'N'$ , it is  $B_{ib:ib+k-1, jb:jb+n-1}$ .
  - For  $transb = 'T'$  or  $'C'$ , it is  $B_{ib:ib+n-1, jb:jb+k-1}$ .
- $C$  represents the global general submatrix  $C_{ic:ic+m-1, jc:jc+n-1}$ .
- $\alpha$  and  $\beta$  are scalars.

**Note:** No data should be moved to form  $A^T$ ,  $A^H$ ,  $B^T$ , or  $B^H$ ; that is, the  $A$  and  $B$  matrices should always be stored in their untransposed forms.

In the following four cases, no computation is performed and the subroutine returns after doing some parameter checking:

- $m = 0$
- $n = 0$
- $\alpha$  is zero and  $\beta$  is one.
- $k = 0$  and  $\beta$  is one.

Assuming the above conditions do not exist, if  $\beta$  is not one and  $k$  is 0, then  $\beta C$  is returned.

See references [15] and [16].

Table 57. Data Types

$A, B, C, \alpha, \beta$	Subroutine
Long-precision real	PDGEMM
Long-precision complex	PZGEMM

## Syntax

<b>Fortran</b>	CALL PDGEMM   PZGEMM ( <i>transa</i> , <i>transb</i> , <i>m</i> , <i>n</i> , <i>k</i> , <i>alpha</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>b</i> , <i>ib</i> , <i>jb</i> , <i>desc_b</i> , <i>beta</i> , <i>c</i> , <i>ic</i> , <i>jc</i> , <i>desc_c</i> )
<b>C and C++</b>	pdgemm   pzgemm ( <i>transa</i> , <i>transb</i> , <i>m</i> , <i>n</i> , <i>k</i> , <i>alpha</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>b</i> , <i>ib</i> , <i>jb</i> , <i>desc_b</i> , <i>beta</i> , <i>c</i> , <i>ic</i> , <i>jc</i> , <i>desc_c</i> );

### On Entry

*transa* indicates the form of matrix *A* to use in the computation, where:

If *transa* = 'N', *A* is used in the computation.

If *transa* = 'T',  $A^T$  is used in the computation.

If *transa* = 'C',  $A^H$  is used in the computation.

Scope: **global**

Specified as: a single character; *transa* = 'N', 'T', or 'C'

*transb* indicates the form of matrix *B* to use in the computation, where:

If *transb* = 'N', *B* is used in the computation.

If *transb* = 'T',  $B^T$  is used in the computation.

If *transb* = 'C',  $B^H$  is used in the computation.

Scope: **global**

Specified as: a single character; *transb* = 'N', 'T', or 'C'

*m* is the number of rows in submatrix *C* used in the computation, and:

If *transa* = 'N', it is the number of rows in submatrix *A*.

If *transa* = 'T' or 'C', it is the number of columns in submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $m \geq 0$ .

*n* is the number of columns in submatrix *C* used in the computation, and:

If *transb* = 'N', it is the number of columns in submatrix *B*.

If *transb* = 'T' or 'C', it is the number of rows in submatrix *B*.

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

*k* has the following meaning:

If *transa* = 'N', it is the number of columns in submatrix *A*.

If *transa* = 'T' or 'C', it is the number of rows in submatrix *A*.

In addition:

If *transb* = 'N', it is the number of rows in submatrix *B*.

If *transb* = 'T' or 'C', it is the number of columns in submatrix *B*.

Scope: **global**

Specified as: a fullword integer;  $k \geq 0$ .

*alpha* is the scalar  $\alpha$ .

Scope: **global**

Specified as: a number of the data type indicated in Table 57 on page 239.

- a* is the local part of the global general matrix *A*. This identifies the **first element** of the local array *A*. This subroutine computes the location of the first element of the local subarray used, based on *ia*, *ja*, *desc\_a*, *p*, *q*, *myrow*, and *mycol*; therefore:
- If *transa* = 'N', the leading LOCp(*ia+m-1*) by LOCq(*ja+k-1*) part of the local array *A* must contain the local pieces of the leading *ia+m-1* by *ja+k-1* part of the global matrix.
  - If *transa* = 'T' or 'C', the leading LOCp(*ia+k-1*) by LOCq(*ja+m-1*) part of the local array *A* must contain the local pieces of the leading *ia+k-1* by *ja+m-1* part of the global matrix.

**Note:** No data should be moved to form  $A^T$  or  $A^H$ ; that is, the matrix *A* should always be stored in its untransposed form.

Scope: **local**

Specified as: an LLD\_ *A* by (at least) LOCq(*N\_A*) array, containing numbers of the data type indicated in Table 57 on page 239. Details about the block-cyclic data distribution of global matrix *A* are stored in *desc\_a*.

- ia* is the row index of the global matrix *A*, identifying the first row of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq M\_A$ , and:

If *transa* = 'N', then  $ia+m-1 \leq M\_A$ .

If *transa* = 'T' or 'C', then  $ia+k-1 \leq M\_A$ .

- ja* is the column index of the global matrix *A*, identifying the first column of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$ , and:

If *transa* = 'N', then  $ja+k-1 \leq N\_A$ .

If *transa* = 'T' or 'C', then  $ja+m-1 \leq N\_A$ .

*desc\_a* is the array descriptor for global matrix *A*, described in the following table:

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $m = 0$ or $k = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	If $m = 0$ or $k = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global

## PDGEMM and PZGEMM

<i>desc_a</i>	Name	Description	Limits	Scope
6	NB_A	Column block size	$NB\_A \geq 1$	Global
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_A < p$	Global
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < q$	Global
9	LLD_A	The leading dimension of the local array	$LLD\_A \geq \max(1, LOCp(M\_A))$	Local

Specified as: an array of (at least) length 9, containing fullword integers.

*b* is the local part of the global general matrix *B*. This identifies the **first element** of the local array *B*. This subroutine computes the location of the first element of the local subarray used, based on *ib*, *jb*, *desc\_b*, *p*, *q*, *myrow*, and *mycol*; therefore:

- If *transb* = 'N', the leading  $LOCp(ib+k-1)$  by  $LOCq(jb+n-1)$  part of the local array *B* must contain the local pieces of the leading  $ib+k-1$  by  $jb+n-1$  part of the global matrix.
- If *transb* = 'T' or 'C', the leading  $LOCp(ib+n-1)$  by  $LOCq(jb+k-1)$  part of the local array *B* must contain the local pieces of the leading  $ib+n-1$  by  $jb+k-1$  part of the global matrix.

**Note:** No data should be moved to form  $B^T$  or  $B^H$ ; that is, the matrix *B* should always be stored in its untransposed form.

Scope: **local**

Specified as: an  $LLD\_B$  by (at least)  $LOCq(N\_B)$  array, containing numbers of the data type indicated in Table 57 on page 239. Details about the block-cyclic data distribution of global matrix *B* are stored in *desc\_b*.

*ib* is the row index of the global matrix *B*, identifying the first row of the submatrix *B*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ib \leq M\_B$ , and:

If *transb* = 'N', then  $ib+k-1 \leq M\_B$ .

If *transb* = 'T' or 'C', then  $ib+n-1 \leq M\_B$ .

*jb* is the column index of the global matrix *B*, identifying the first column of the submatrix *B*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq jb \leq N\_B$ , and:

If *transb* = 'N', then  $jb+n-1 \leq N\_B$ .

If *transb* = 'T' or 'C', then  $jb+k-1 \leq N\_B$ .

*desc\_b* is the array descriptor for global matrix *B*, described in the following table:

<i>desc_b</i>	Name	Description	Limits	Scope
1	DTYPE_B	Descriptor type	$DTYPE\_B=1$	Global

<i>desc_b</i>	Name	Description	Limits	Scope
2	CTXT_B	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_B	Number of rows in the global matrix	If $k = 0$ or $n = 0$ : $M\_B \geq 0$ Otherwise: $M\_B \geq 1$	Global
4	N_B	Number of columns in the global matrix	If $k = 0$ or $n = 0$ : $N\_B \geq 0$ Otherwise: $N\_B \geq 1$	Global
5	MB_B	Row block size	$MB\_B \geq 1$	Global
6	NB_B	Column block size	$NB\_B \geq 1$	Global
7	RSRC_B	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_B < p$	Global
8	CSRC_B	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_B < q$	Global
9	LLD_B	The leading dimension of the local array	$LLD\_B \geq \max(1, LOCp(M\_B))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

*beta* is the scalar  $\beta$ .

Scope: **global**

Specified as: a number of the data type indicated in Table 57 on page 239.

*c* is the local part of the global general matrix **C**. This identifies the **first element** of the local array **C**. This subroutine computes the location of the first element of the local subarray used, based on *ic*, *jc*, *desc\_c*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading  $LOCp(ic+m-1)$  by  $LOCq(jc+n-1)$  part of the local array **C** must contain the local pieces of the leading  $ic+m-1$  by  $jc+n-1$  part of the global matrix.

When  $\beta$  is zero, **C** need not be set on input.

Scope: **local**

Specified as: an  $LLD\_C$  by (at least)  $LOCq(N\_C)$  array, containing numbers of the data type indicated in Table 57 on page 239. Details about the block-cyclic data distribution of global matrix **C** are stored in *desc\_c*.

*ic* is the row index of the global matrix **C**, identifying the first row of the submatrix **C**.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ic \leq M\_C$  and  $ic+m-1 \leq M\_C$ .

*jc* is the column index of the global matrix **C**, identifying the first column of the submatrix **C**.

Scope: **global**

Specified as: a fullword integer;  $1 \leq jc \leq N\_C$  and  $jc+n-1 \leq N\_C$ .

## PDGEMM and PZGEMM

*desc\_c* is the array descriptor for global matrix *C*, described in the following table:

<i>desc_c</i>	Name	Description	Limits	Scope
1	DTYPE_C	Descriptor type	DTYPE_C=1	Global
2	CTXT_C	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_C	Number of rows in the global matrix	If $m = 0$ or $n = 0$ : $M\_C \geq 0$ Otherwise: $M\_C \geq 1$	Global
4	N_C	Number of columns in the global matrix	If $m = 0$ or $n = 0$ : $N\_C \geq 0$ Otherwise: $N\_C \geq 1$	Global
5	MB_C	Row block size	$MB\_C \geq 1$	Global
6	NB_C	Column block size	$NB\_C \geq 1$	Global
7	RSRC_C	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_C < p$	Global
8	CSRC_C	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_C < q$	Global
9	LLD_C	The leading dimension of the local array	$LLD\_C \geq \max(1, LOCp(M\_C))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

### On Return

*c* is the updated local part of the global matrix *C*, containing the results of the computation.

Scope: **local**

Returned as: an LLD\_C by (at least) LOCq(N\_C) array, containing numbers of the data type indicated in Table 57 on page 239.

## Notes and Coding Rules

- These subroutines accept lowercase letters for the *transa* and *transb* arguments.
- For PDGEMM, if you specify 'C' for the *transa* or *transb* argument, it is interpreted as though you specified 'T'.
- The matrices must have no common elements; otherwise, results are unpredictable.
- The NUMROC utility subroutine can be used to determine the values of LOCp(M\_) and LOCq(N\_) used in the argument descriptions above. For details, see "Determining the Number of Rows and Columns in Your Local Arrays" on page 28 and "NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process" on page 868.
- For suggested block sizes, see "Coding Tips for Optimizing Parallel Performance" on page 77.
- The following values must be equal: CTXT\_A = CTXT\_B = CTXT\_C.

7. The coding rules described in this note depend upon which matrix— $A$ ,  $B$ , or  $C$ —is used as the reference matrix, which is referred to, in general, as matrix  $X$ . For each of the three possible selections for the reference matrix, there is a unique set of coding rules that must be met. These are detailed in Table 58 and Table 59 on page 246. Follow these steps to select a reference matrix and determine what coding rules to use:

**Step 1:** First, the reference matrix is selected. For optimal performance, the reference matrix is selected based on the arguments  $m$ ,  $n$ , and  $k$ , as follows:

- If  $k \leq \min(m, n)$ , then  $X = C$
- If  $n \leq \min(m, k)$ , then  $X = A$
- If  $m \leq \min(n, k)$ , then  $X = B$

The matrix selected must satisfy **coding rules a and d**, described below, to be a suitable reference matrix. If it does, you go to step 2. If it does not, then it checks to see if either of the other two matrices satisfies **coding rules a, c, and d**, making one of them a suitable reference matrix. If one of them is suitable, then you go to step 2. If neither matrix is suitable, an error condition results.

**Step 2:** After a suitable reference matrix is chosen in Step 2, **all remaining coding rules**, described below, are checked. If the rules are satisfied, the subroutine continues normally. If they are not, an error condition results.

**Coding Rules:** Following are the coding rules:

- The reference matrix must be aligned on a block boundary; that is:
  - $ix-1$  must be a multiple of  $MB\_X$ .
  - $jx-1$  must be a multiple of  $NB\_X$ .

These indexes are indicated in column 5 of Table 58 for each entry for  $X$ .
- The block sizes that must be equal are indicated in column 4 of Table 58 for each entry for  $X$ . The rules for block sizes depend only upon the values of  $transa$  and  $transb$ , and not on the reference matrix selected; however, for your convenience, the rules are repeated in the table for each reference matrix.
- Given the reference matrix  $X$ , additional rules apply to the block row and block column offsets of the two nonreference matrices. These rules are listed in column 7 of Table 58 for each entry for  $X$ . These rules must only be met when looping is required—that is, either of the conditions in column 8 is met.
- The indexes of the nonreference matrices, which need to be on a block boundary, are listed in column 6 of Table 58 for each entry for  $X$ .

Table 58. Coding Rules for the Reference Matrix  $X$

-1- X	-2- <i>transa</i>	-3- <i>transb</i>	-4- (b) Equal Block Sizes	-5- (a) Block Bndry For X	-6- (d) Block Bndry For Other	-7- (c) Equal Block Offsets (If Looping is Required)	-8- (c) Conditions For Looping
A	'N'	'N'	MB_A = MB_C NB_B = NB_C NB_A = MB_B	<i>ia, ja</i>	<i>ib, ic</i>	$\text{mod}(jb-1, NB\_B)$ = $\text{mod}(jc-1, NB\_C)$	$n+\text{mod}(jb-1, NB\_B) > NB\_B$ -or- $n+\text{mod}(jc-1, NB\_C) > NB\_C$
A	'N'	'T' or 'C'	MB_A = MB_C MB_B = NB_C NB_A = NB_B	<i>ia, ja</i>	<i>jb, ic</i>	$\text{mod}(ib-1, MB\_B)$ = $\text{mod}(jc-1, NB\_C)$	$n+\text{mod}(ib-1, MB\_B) > MB\_B$ -or- $n+\text{mod}(jc-1, NB\_C) > NB\_C$
A	'T' or 'C'	'N'	NB_A = MB_C NB_B = NB_C MB_A = MB_B	<i>ia, ja</i>	<i>ib, ic</i>	$\text{mod}(jb-1, NB\_B)$ = $\text{mod}(jc-1, NB\_C)$	$n+\text{mod}(jb-1, NB\_B) > NB\_B$ -or- $n+\text{mod}(jc-1, NB\_C) > NB\_C$

Table 58. Coding Rules for the Reference Matrix  $X$  (continued)

-1- $X$	-2- <i>transa</i>	-3- <i>transb</i>	-4- (b) Equal Block Sizes	-5- (a) Block Bndry For $X$	-6- (d) Block Bndry For Other	-7- (c) Equal Block Offsets (If Looping is Required)	-8- (c) Conditions For Looping
$A$	'T' or 'C'	'T' or 'C'	NB_A = MB_C MB_B = NB_C MB_A = NB_B	$ia, ja$	$jb, ic$	$\text{mod}(ib-1, MB_B)$ = $\text{mod}(jc-1, NB_C)$	$n+\text{mod}(ib-1, MB_B) > MB_B$ -or- $n+\text{mod}(jc-1, NB_C) > NB_C$
$B$	'N'	'N'	MB_A = MB_C NB_B = NB_C NB_A = MB_B	$ib, jb$	$ja, jc$	$\text{mod}(ia-1, MB_A)$ = $\text{mod}(ic-1, MB_C)$	$m+\text{mod}(ia-1, MB_A) > MB_A$ -or- $m+\text{mod}(ic-1, MB_C) > MB_C$
$B$	'N'	'T' or 'C'	MB_A = MB_C MB_B = NB_C NB_A = NB_B	$ib, jb$	$ja, jc$	$\text{mod}(ia-1, MB_A)$ = $\text{mod}(ic-1, MB_C)$	$m+\text{mod}(ia-1, MB_A) > MB_A$ -or- $m+\text{mod}(ic-1, MB_C) > MB_C$
$B$	'T' or 'C'	'N'	NB_A = MB_C NB_B = NB_C MB_A = MB_B	$ib, jb$	$ia, jc$	$\text{mod}(ja-1, NB_A)$ = $\text{mod}(ic-1, MB_C)$	$m+\text{mod}(ja-1, NB_A) > NB_A$ -or- $m+\text{mod}(ic-1, MB_C) > MB_C$
$B$	'T' or 'C'	'T' or 'C'	NB_A = MB_C MB_B = NB_C MB_A = NB_B	$ib, jb$	$ia, jc$	$\text{mod}(ja-1, NB_A)$ = $\text{mod}(ic-1, MB_C)$	$m+\text{mod}(ja-1, NB_A) > NB_A$ -or- $m+\text{mod}(ic-1, MB_C) > MB_C$
$C$	'N'	'N'	MB_A = MB_C NB_B = NB_C NB_A = MB_B	$ic, jc$	$ia, jb$	$\text{mod}(ja-1, NB_A)$ = $\text{mod}(ib-1, MB_B)$	$k+\text{mod}(ja-1, NB_A) > NB_A$ -or- $k+\text{mod}(ib-1, MB_B) > MB_B$
$C$	'N'	'T' or 'C'	MB_A = MB_C MB_B = NB_C NB_A = NB_B	$ic, jc$	$ia, ib$	$\text{mod}(ja-1, NB_A)$ = $\text{mod}(jb-1, NB_B)$	$k+\text{mod}(ja-1, NB_A) > NB_A$ -or- $k+\text{mod}(jb-1, NB_B) > NB_B$
$C$	'T' or 'C'	'N'	NB_A = MB_C NB_B = NB_C MB_A = MB_B	$ic, jc$	$ja, jb$	$\text{mod}(ia-1, MB_A)$ = $\text{mod}(ib-1, MB_B)$	$k+\text{mod}(ia-1, MB_A) > MB_A$ -or- $k+\text{mod}(ib-1, MB_B) > MB_B$
$C$	'T' or 'C'	'T' or 'C'	NB_A = MB_C MB_B = NB_C MB_A = NB_B	$ic, jc$	$ja, ib$	$\text{mod}(ia-1, MB_A)$ = $\text{mod}(jb-1, NB_B)$	$k+\text{mod}(ia-1, MB_A) > MB_A$ -or- $k+\text{mod}(jb-1, NB_B) > NB_B$

- e. Additional rules apply to the row and column alignment of the various matrices in the process grid; specifically, the process row or process column containing the first row or column of the reference submatrix  $X$ , respectively, must also contain the first row or column of one of the other two nonreference submatrices, as indicated in column 4 of Table 59 for each entry for  $X$ . Following is the definition of  $ixrow$  and  $ixcol$ , which holds true for  $A$ ,  $B$ , and  $C$ :

- $ixrow = \text{mod}((((ix-1)/MB_X)+RSRC_X), p)$
- $ixcol = \text{mod}((((jx-1)/NB_X)+CSRC_X), q)$

Table 59. Coding Rules for the Reference Matrix  $X$

-1- $X$	-2- <i>transa</i>	-3- <i>transb</i>	-4- (e) Process Grid Alignment
$A$	'N'	'N'	$iarow = icrow$
$A$	'N'	'T' or 'C'	$iarow = icrow$ $ibcol = iacol$



Table 59. Coding Rules for the Reference Matrix *X* (continued)

-1- <i>X</i>	-2- <i>transa</i>	-3- <i>transb</i>	-4- (e) Process Grid Alignment
<i>A</i>	'T' or 'C'	'N'	<i>iarow</i> = <i>ibrow</i>
<i>A</i>	'T' or 'C'	'T' or 'C'	(no rules)
<i>B</i>	'N'	'N'	<i>ibcol</i> = <i>iccol</i>
<i>B</i>	'N'	'T' or 'C'	<i>ibcol</i> = <i>iacol</i>
<i>B</i>	'T' or 'C'	'N'	<i>iarow</i> = <i>ibrow</i> <i>ibcol</i> = <i>iccol</i>
<i>B</i>	'T' or 'C'	'T' or 'C'	(no rules)
<i>C</i>	'N'	'N'	<i>iarow</i> = <i>icrow</i> <i>ibcol</i> = <i>iccol</i>
<i>C</i>	'N'	'T' or 'C'	<i>iarow</i> = <i>icrow</i>
<i>C</i>	'T' or 'C'	'N'	<i>ibcol</i> = <i>iccol</i>
<i>C</i>	'T' or 'C'	'T' or 'C'	(no rules)

**Example:** Following is an example of the coding rules necessary for the case where *transa* = 'N' and *transb* = 'N', where the reference matrix selected is *A*. Following are the indexes, dimensions, and block sizes used in the computation for the matrices:

Indexes:             $\begin{array}{cc} ic & jc \\ | & | \end{array}$              $\begin{array}{cc} ia & ja \\ | & | \end{array}$              $\begin{array}{cc} ib & jb \\ | & | \end{array}$              $\begin{array}{cc} ic & jc \\ | & | \end{array}$

Dimensions:  $C(m, n)$   
 $\leftarrow \alpha A(m, k) B(k, n) + \beta C(m, n)$

Block Sizes:     $\begin{array}{cc} MB\_C & NB\_C \\ | & | \end{array}$              $\begin{array}{cc} MB\_A & NB\_A \\ | & | \end{array}$              $\begin{array}{cc} MB\_B & NB\_B \\ | & | \end{array}$              $\begin{array}{cc} MB\_C & NB\_C \\ | & | \end{array}$

- A* must be aligned on a block boundary, as indicated in column 5 in Table 58 on page 245:
  - ia*-1 must be a multiple of MB\_A.
  - ja*-1 must be a multiple of NB\_A.
- The block sizes that correspond to each matrix dimension must be equal, where MB\_ represents the row dimension and NB\_ represents the column dimension, as indicated in column 4 in Table 58 on page 245:
  - MB\_A = MB\_C
  - NB\_B = NB\_C
  - NB\_A = MB\_B
- As shown above, *m* and *k* are the dimensions of the reference matrix *A*; therefore, *n* is used to determine if looping is required; that is, if one of the following is true, as indicated in column 8 in Table 58 on page 245:
  - $n + \text{mod}(jc-1, NB\_C) > NB\_C$
  - $n + \text{mod}(jb-1, NB\_B) > NB\_B$
 then the following offsets must be equal, as indicated in column 7 in Table 58 on page 245:
  - $\text{mod}(jb-1, NB\_B) = \text{mod}(jc-1, NB\_C)$
- The other indexes from each of the nonreference matrices—not used in c above—must be aligned on a block boundary, as indicated in column 6 in Table 58 on page 245:
  - ic*-1 must be a multiple of MB\_C.
  - ib*-1 must be a multiple of MB\_B.

- e. In the process grid, the process row containing the first row of the submatrix  $A$  must also contain the first row of the submatrix  $C$ , as indicated in column 4 in Table 59 on page 246; that is,  $iarrow = icrow$ , where:
  - $iarrow = \text{mod}(((ia-1)/MB\_A)+RSRC\_A), p)$
  - $icrow = \text{mod}(((ic-1)/MB\_C)+RSRC\_C), p)$

## Error Conditions

### Computational Errors

None

### Resource Errors

Unable to allocate work space

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. DTYPE\_A is invalid.
2. DTYPE\_B is invalid.
3. DTYPE\_C is invalid.

#### Stage 2:

1. CTXT\_A is invalid.

#### Stage 3:

1. The subroutine was called from outside the process grid.

#### Stage 4:

1.  $transa \neq 'N', 'T', \text{ or } 'C'$
2.  $transb \neq 'N', 'T', \text{ or } 'C'$
3.  $m < 0$
4.  $n < 0$
5.  $k < 0$
6.  $M\_A < 0$  and  $(m = 0 \text{ or } k = 0)$ ;  $M\_A < 1$  otherwise
7.  $N\_A < 0$  and  $(m = 0 \text{ or } k = 0)$ ;  $N\_A < 1$  otherwise
8.  $M\_B < 0$  and  $(k = 0 \text{ or } n = 0)$ ;  $M\_B < 1$  otherwise
9.  $N\_B < 0$  and  $(k = 0 \text{ or } n = 0)$ ;  $N\_B < 1$  otherwise
10.  $M\_C < 0$  and  $(m = 0 \text{ or } n = 0)$ ;  $M\_C < 1$  otherwise
11.  $N\_C < 0$  and  $(m = 0 \text{ or } n = 0)$ ;  $N\_C < 1$  otherwise
12.  $ia < 1$
13.  $ib < 1$
14.  $ic < 1$
15.  $ja < 1$
16.  $jb < 1$
17.  $jc < 1$
18.  $MB\_A < 1$
19.  $MB\_B < 1$
20.  $MB\_C < 1$
21.  $NB\_A < 1$
22.  $NB\_B < 1$
23.  $NB\_C < 1$
24.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
25.  $RSRC\_B < 0$  or  $RSRC\_B \geq p$
26.  $RSRC\_C < 0$  or  $RSRC\_C \geq p$
27.  $CSRC\_A < 0$  or  $CSRC\_A \geq q$
28.  $CSRC\_B < 0$  or  $CSRC\_B \geq q$

29.  $CSRC\_C < 0$  or  $CSRC\_C \geq q$
30.  $CTXT\_A \neq CTXT\_B$
31.  $CTXT\_A \neq CTXT\_C$

**Stage 5:** If  $m \neq 0$  and  $k \neq 0$ :

1.  $transa = 'N'$  and  $ia+m-1 > M\_A$
2.  $transa = 'T'$  or  $'C'$  and  $ia+k-1 > M\_A$
3.  $transa = 'N'$  and  $ja+k-1 > N\_A$
4.  $transa = 'T'$  or  $'C'$  and  $ja+m-1 > N\_A$
5.  $ia > M\_A$
6.  $ja > N\_A$
- If  $n \neq 0$  and  $k \neq 0$ :
  7.  $transb = 'N'$  and  $ib+k-1 > M\_B$
  8.  $transb = 'T'$  or  $'C'$  and  $ib+n-1 > M\_B$
  9.  $transb = 'N'$  and  $jb+n-1 > N\_B$
  10.  $transb = 'T'$  or  $'C'$  and  $jb+k-1 > N\_B$
  11.  $ib > M\_B$
  12.  $jb > N\_B$
- If  $m \neq 0$  and  $n \neq 0$ :
  13.  $ic+m-1 > M\_C$
  14.  $jc+n-1 > N\_C$
  15.  $ic > M\_C$
  16.  $jc > N\_C$
17. For the reference matrix (defined in note 7 in “Notes and Coding Rules” on page 244) and the appropriate  $transa$  and  $transb$  values, the indexes listed in column 5 of Table 58 are not aligned on a block boundary, where boundary alignment is defined as:
  - $ix-1$  must be a multiple of  $MB\_X$ .
  - $jx-1$  must be a multiple of  $NB\_X$ .
18. For the two nonreference matrices (defined in note 7 in “Notes and Coding Rules” on page 244) and the appropriate  $transa$  and  $transb$  values, the indexes listed in column 6 of Table 58 are not aligned on a block boundary. Using  $Z$  to represent one of the nonreference matrices, each boundary alignment is expressed as one of the following:
  - $iz-1$  must be a multiple of  $MB\_Z$ .
  - $jz-1$  must be a multiple of  $NB\_Z$ .
19. For the reference matrix (defined in note 7 in “Notes and Coding Rules” on page 244) and the appropriate  $transa$  and  $transb$  values, if looping occurs—that is, one of the conditions in column 8 of Table 58 is true—then the block offsets indicated in column 7 are not equal.

**Stage 6:**

1. For the appropriate  $transa$  and  $transb$  values indicated in Table 58 (where the reference matrix does not matter), some of the block sizes indicated in column 4 are not equal.
2.  $LLD\_A < \max(1, LOCp(M\_A))$
3.  $LLD\_B < \max(1, LOCp(M\_B))$
4.  $LLD\_C < \max(1, LOCp(M\_C))$
5. In the process grid, the process row or process column containing the first row or column of the reference submatrix  $X$  (defined in note 7 in “Notes and Coding Rules” on page 244), respectively, does not contain the first row or column of one of the other two nonreference submatrices, as indicated in column 4 of Table 59. Following is the definition of  $ixrow$  and  $ixcol$ , which holds true for  $A$ ,  $B$ , and  $C$ :
  - $ixrow = \text{mod}((((ix-1)/MB\_X)+RSRC\_X), p)$
  - $ixcol = \text{mod}((((jx-1)/NB\_X)+CSRC\_X), q)$

## Examples

### Example 1

This example computes  $C = \beta C + \alpha AB$  using a  $2 \times 2$  process grid.

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      TRANSA  TRANSB  M      N      K  ALPHA      A  IA  JA  DESC_A  B  IB  JB
      |      |      |      |      |      |      |  |  |  |      |  |  |
CALL PDGEMM( 'N' , 'N' , 6 , 4 , 5 , 1.0D0 , A , 1 , 1 , DESC_A , B , 1 , 1 ,
      DESC_B  BETA  C  IC  JC  DESC_C
      |      |      |  |  |  |
      DESC_B , 2.0D0 , C , 1 , 1 , DESC_C )
```

	Desc_A	Desc_B	Desc_C
DTYPE_	1	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	6	5	6
N_	5	4	4
MB_	3	2	3
NB_	2	2	2
RSRC_	0	0	0
CSRC_	0	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>	See below <sup>2</sup>

#### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:

```
LLD_A = MAX(1, NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_B = MAX(1, NUMROC(M_B, MB_B, MYROW, RSRC_B, NPROW))
LLD_C = MAX(1, NUMROC(M_C, MB_C, MYROW, RSRC_C, NPROW))
```

In this example, LLD\_A = LLD\_C = 3 on all processes, and LLD\_B = 3 on P<sub>10</sub> and P<sub>01</sub> and LLD\_B = 2 on P<sub>11</sub> and P<sub>00</sub>.

Global general  $6 \times 5$  matrix *A* with block size  $3 \times 2$ :

B,D	0	1	2
0	$\begin{bmatrix} 1.0 & 2.0 \\ 2.0 & 0.0 \\ 1.0 & -1.0 \end{bmatrix}$	$\begin{bmatrix} -1.0 & -1.0 \\ 1.0 & 1.0 \\ -1.0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 4.0 \\ -1.0 \\ 2.0 \end{bmatrix}$
1	$\begin{bmatrix} -3.0 & 2.0 \\ 4.0 & 0.0 \\ -1.0 & -1.0 \end{bmatrix}$	$\begin{bmatrix} 2.0 & 2.0 \\ -2.0 & 1.0 \\ 1.0 & -3.0 \end{bmatrix}$	$\begin{bmatrix} 0.0 \\ -1.0 \\ 2.0 \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$

Local arrays for  $A$ :

p,q	0	1
0	1.0 2.0 4.0 2.0 0.0 -1.0 1.0 -1.0 2.0	-1.0 -1.0 1.0 1.0 -1.0 1.0
1	-3.0 2.0 0.0 4.0 0.0 -1.0 -1.0 -1.0 2.0	2.0 2.0 -2.0 1.0 1.0 -3.0

Global general  $5 \times 4$  matrix  $B$  with block size  $2 \times 2$ :

B,D	0	1
0	1.0 -1.0 2.0 2.0	0.0 2.0 -1.0 -2.0
1	1.0 0.0 -3.0 -1.0	-1.0 1.0 1.0 -1.0
2	4.0 2.0	-1.0 1.0

The following is the  $2 \times 2$  process grid:

B,D	0	1
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

Local arrays for  $B$ :

p,q	0	1
0	1.0 -1.0 2.0 2.0 4.0 2.0	0.0 2.0 -1.0 -2.0 -1.0 1.0
1	1.0 0.0 -3.0 -1.0	-1.0 1.0 1.0 -1.0

Global general  $6 \times 4$  matrix  $C$  with block size  $3 \times 2$ :

B,D	0	1
0	0.5 0.5 0.5 0.5 0.5 0.5	0.5 0.5 0.5 0.5 0.5 0.5
1	0.5 0.5 0.5 0.5 0.5 0.5	0.5 0.5 0.5 0.5 0.5 0.5

The following is the  $2 \times 2$  process grid:

## PDGEMM and PZGEMM

B,D	0	1
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$

Local arrays for  $C$ :

p,q	0	1
0	0.5 0.5 0.5 0.5 0.5 0.5	0.5 0.5 0.5 0.5 0.5 0.5
1	0.5 0.5 0.5 0.5 0.5 0.5	0.5 0.5 0.5 0.5 0.5 0.5

**Output:**

Global general  $6 \times 4$  matrix  $C$  with block size  $3 \times 2$ :

B,D	0	1
0	$\begin{bmatrix} 24.0 & 13.0 \\ -3.0 & -4.0 \\ 4.0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} -5.0 & 3.0 \\ 2.0 & 4.0 \\ 2.0 & 5.0 \end{bmatrix}$
1	$\begin{bmatrix} -2.0 & 6.0 \\ -4.0 & -6.0 \\ 16.0 & 7.0 \end{bmatrix}$	$\begin{bmatrix} -1.0 & -9.0 \\ 5.0 & 5.0 \\ -4.0 & 7.0 \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0	1
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$

Local arrays for  $C$ :

p,q	0	1
0	24.0 13.0 -3.0 -4.0 4.0 1.0	-5.0 3.0 2.0 4.0 2.0 5.0
1	-2.0 6.0 -4.0 -6.0 16.0 7.0	-1.0 -9.0 5.0 5.0 -4.0 7.0

### Example 2

This example computes  $C = \beta C + \alpha AB$  using a  $2 \times 2$  process grid.

## Call Statements and Input:

```

ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      TRANSA  TRANSB  M      N      K      ALPHA      A  IA  JA  DESC_A  B  IB  JB
      |      |      |      |      |      |      |  |  |  |  |      |  |  |
CALL PZGEMM('N' , 'N' , 6 , 2 , 3 , (1.0D0,0.0D0) , A , 1 , 1 , DESC_A , B , 1 , 1 ,

      DESC_B      BETA      C  IC  JC  DESC_C
      |      |      |      |  |  |  |
DESC_B , (2.0D0,0.0D0) , C , 1 , 1 , DESC_C)

```

	Desc_A	Desc_B	Desc_C
DTYPE_	1	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	6	3	6
N_	3	2	2
MB_	2	2	2
NB_	2	2	2
RSRC_	0	0	0
CSRC_	0	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>	See below <sup>2</sup>

## Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:

```

LLD_A = MAX(1,NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_B = MAX(1,NUMROC(M_B, MB_B, MYROW, RSRC_B, NPROW))
LLD_C = MAX(1,NUMROC(M_C, MB_C, MYROW, RSRC_C, NPROW))

```

In this example, LLD\_A = 4 on P<sub>00</sub> and P<sub>01</sub> and LLD\_A = 2 on P<sub>10</sub> and P<sub>11</sub>.  
 LLD\_B = 2 on P<sub>00</sub> and LLD\_B = 1 on P<sub>10</sub>. LLD\_C = 4 on P<sub>00</sub> and LLD\_C = 2 on P<sub>10</sub>.

Global general  $6 \times 3$  matrix *A* with block size  $2 \times 2$ :

$$\begin{array}{c} \text{B,D} \end{array} \quad \begin{array}{cc} 0 & 1 \end{array}$$

$$\begin{array}{c} 0 \\ 1 \\ 2 \end{array} \left[ \begin{array}{cc|cc} (1.0,5.0) & (9.0,2.0) & (1.0,9.0) & \\ (2.0,4.0) & (8.0,3.0) & (1.0,8.0) & \\ \hline (3.0,3.0) & (7.0,5.0) & (1.0,7.0) & \\ (4.0,2.0) & (4.0,7.0) & (1.0,5.0) & \\ \hline (5.0,1.0) & (5.0,1.0) & (1.0,6.0) & \\ (6.0,6.0) & (3.0,6.0) & (1.0,4.0) & \end{array} \right]$$

The following is the  $2 \times 2$  process grid:

## PDGEMM and PZGEMM

B,D	0	1
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

Local arrays for  $A$ :

p,q	0	1
0	$\begin{pmatrix} (1.0,5.0) & (9.0,2.0) \\ (2.0,4.0) & (8.0,3.0) \\ (5.0,1.0) & (5.0,1.0) \\ (6.0,6.0) & (3.0,6.0) \end{pmatrix}$	$\begin{pmatrix} (1.0,9.0) \\ (1.0,8.0) \\ (1.0,6.0) \\ (1.0,4.0) \end{pmatrix}$
1	$\begin{pmatrix} (3.0,3.0) & (7.0,5.0) \\ (4.0,2.0) & (4.0,7.0) \end{pmatrix}$	$\begin{pmatrix} (1.0,7.0) \\ (1.0,5.0) \end{pmatrix}$

Global general  $3 \times 2$  matrix  $B$  with block size  $2 \times 2$ :

B,D	0
0	$\begin{pmatrix} (1.0,8.0) & (2.0,7.0) \\ (4.0,4.0) & (6.0,8.0) \end{pmatrix}$
1	$\begin{pmatrix} (6.0,2.0) & (4.0,5.0) \end{pmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0	--
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$

Local arrays for  $B$ :

p,q	0
0	$\begin{pmatrix} (1.0,8.0) & (2.0,7.0) \\ (4.0,4.0) & (6.0,8.0) \end{pmatrix}$
1	$\begin{pmatrix} (6.0,2.0) & (4.0,5.0) \end{pmatrix}$

Global general  $6 \times 2$  matrix  $C$  with block size  $2 \times 2$ :

B,D	0
0	$\begin{pmatrix} (0.5,0.0) & (0.5,0.0) \\ (0.5,0.0) & (0.5,0.0) \end{pmatrix}$
1	$\begin{pmatrix} (0.5,0.0) & (0.5,0.0) \\ (0.5,0.0) & (0.5,0.0) \end{pmatrix}$
2	$\begin{pmatrix} (0.5,0.0) & (0.5,0.0) \\ (0.5,0.0) & (0.5,0.0) \end{pmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0	--
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$



Local arrays for  $C$ :

p,q	$\emptyset$	
0	(0.5,0.0)	(0.5,0.0)
	(0.5,0.0)	(0.5,0.0)
	(0.5,0.0)	(0.5,0.0)
	(0.5,0.0)	(0.5,0.0)
1	(0.5,0.0)	(0.5,0.0)
	(0.5,0.0)	(0.5,0.0)

**Output:**

Global general  $6 \times 2$  matrix  $C$  with block size  $2 \times 2$ :

B,D	$\emptyset$	
0	(-22.0,113.0)	(-35.0,142.0)
	(-19.0,114.0)	(-35.0,141.0)
1	(-20.0,119.0)	(-43.0,146.0)
	(-27.0,110.0)	(-58.0,131.0)
2	(8.0,103.0)	(0.0,112.0)
	(-55.0,116.0)	(-75.0,135.0)

The following is the  $2 \times 2$  process grid:

B,D	$\emptyset$	--
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$

Local arrays for  $C$ :

p,q	$\emptyset$	
0	(-22.0,113.0)	(-35.0,142.0)
	(-19.0,114.0)	(-35.0,141.0)
	(8.0,103.0)	(0.0,112.0)
	(-55.0,116.0)	(-75.0,135.0)
1	(-20.0,119.0)	(-43.0,146.0)
	(-27.0,110.0)	(-58.0,131.0)

## PDSYMM, PZSYMM, and PZHEMM — Matrix-Matrix Product Where One Matrix is Real or Complex Symmetric or Complex Hermitian

### Purpose

These subroutines compute one of the following matrix-matrix products:

- 1.  $C \leftarrow \alpha AB + \beta C$
- 2.  $C \leftarrow \alpha BA + \beta C$

where, in the formulas above:

- $A$  represents the global submatrix:
  - For  $side = 'L'$ , it is  $A_{ia:ia+m-1, ja:ja+m-1}$ .
  - For  $side = 'R'$ , it is  $A_{ia:ia+n-1, ja:ja+n-1}$ .
- $B$  represents the global general submatrix  $B_{ib:ib+m-1, jb:jb+n-1}$ .
- $C$  represents the global general submatrix  $C_{ic:ic+m-1, jc:jc+n-1}$ .
- $\alpha$  and  $\beta$  are scalars.

and:

- For PDSYMM, submatrix  $A$  is real symmetric.
- For PZSYMM, submatrix  $A$  is complex symmetric.
- For PZHEMM, submatrix  $A$  is complex Hermitian.

In the following two cases, no computation is performed and the subroutine returns after doing some parameter checking:

- $m = 0$  or  $n = 0$
- $\alpha$  is zero and  $\beta$  is one.

See references [15] and [16].

Table 60. Data Types

$\alpha, \beta, A, B, C$	Subprogram
Long-precision real	PDSYMM
Long-precision complex	PZSYMM and PZHEMM

### Syntax

<b>Fortran</b>	CALL PDSYMM   PZSYMM   PZHEMM ( <i>side</i> , <i>uplo</i> , <i>m</i> , <i>n</i> , <i>alpha</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>b</i> , <i>ib</i> , <i>jb</i> , <i>desc_b</i> , <i>beta</i> , <i>c</i> , <i>ic</i> , <i>jc</i> , <i>desc_c</i> )
<b>C and C++</b>	pdsymm   pzsymm   pzhemm ( <i>side</i> , <i>uplo</i> , <i>m</i> , <i>n</i> , <i>alpha</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>b</i> , <i>ib</i> , <i>jb</i> , <i>desc_b</i> , <i>beta</i> , <i>c</i> , <i>ic</i> , <i>jc</i> , <i>desc_c</i> );

### On Entry

*side* indicates whether  $A$  is located to the left or right of  $B$  in the equation used for this computation, where:

If  $side = 'L'$ ,  $A$  is to the left of  $B$ , resulting in equation 1.

If  $side = 'R'$ ,  $A$  is to the right of  $B$ , resulting in equation 2.

Scope: **global**

Specified as: a single character;  $side = 'L'$  or  $'R'$ .

*uplo* indicates whether the upper or lower triangular part of the global submatrix  $A$  is referenced, where:

If  $uplo = 'U'$ , the upper triangular part is referenced.

If  $uplo = 'L'$ , the lower triangular part is referenced.

Scope: **global**

Specified as: a single character;  $uplo = 'U'$  or  $'L'$ .

$m$  is the number of rows in submatrices  $B$  and  $C$  used in the computation, and:

If  $side = 'L'$ , it is the number of rows and columns in submatrix  $A$  used in the computation.

Scope: **global**

Specified as: a fullword integer;  $m \geq 0$ .

$n$  is the number of columns in submatrices  $B$  and  $C$  used in the computation, and:

If  $side = 'R'$ , it is the number of rows and columns in submatrix  $A$  used in the computation.

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

$alpha$  is the scalar  $\alpha$ .

Scope: **global**

Specified as: a number of the data type indicated in Table 60 on page 256.

$a$  is the local part of the global real symmetric, complex symmetric, or complex Hermitian matrix  $A$ . This identifies the **first element** of the local array  $A$ . This subroutine computes the location of the first element of the local subarray used, based on  $ia$ ,  $ja$ ,  $desc\_a$ ,  $p$ ,  $q$ ,  $myrow$ , and  $mycol$ ; therefore, assuming the following:

If  $side = 'L'$ ,  $numa = m$

If  $side = 'R'$ ,  $numa = n$

the leading  $LOCp(ia+numa-1)$  by  $LOCq(ja+numa-1)$  part of the local array  $A$  must contain the local pieces of the leading  $ia+numa-1$  by  $ja+numa-1$  part of the global matrix, and:

- If  $uplo = 'U'$ , the leading  $numa \times numa$  upper triangular part of the global submatrix  $A_{ia:ia+numa-1, ja:ja+numa-1}$  must contain the upper triangular part of the submatrix, and the strictly lower triangular part is not referenced.
- If  $uplo = 'L'$ , the leading  $numa \times numa$  lower triangular part of the global submatrix  $A_{ia:ia+numa-1, ja:ja+numa-1}$  must contain the lower triangular part of the submatrix, and the strictly upper triangular part is not referenced.

Scope: **local**

Specified as: an  $LLD\_A$  by (at least)  $LOCq(N\_A)$  array, containing numbers of the data type indicated in Table 60 on page 256. Details about the block-cyclic data distribution of global matrix  $A$  are stored in  $desc\_a$ .

$ia$  is the row index of the global matrix  $A$ , identifying the first row of the submatrix  $A$ .

## PDSYMM, PZSYMM, and PZHEMM

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq M\_A$  and  $ia+numa-1 \leq M\_A$ .

*ja* is the column index of the global matrix *A*, identifying the first column of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$  and  $ja+numa-1 \leq N\_A$ .

*desc\_a* is the array descriptor for global matrix *A*, described in the following table:

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $m = 0$ and $side = 'L'$ or $n = 0$ and $side = 'R'$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	If $m = 0$ and $side = 'L'$ or $n = 0$ and $side = 'R'$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_A < p$	Global
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < q$	Global
9	LLD_A	The leading dimension of the local array	$LLD\_A \geq \max(1, LOCp(M\_A))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

*b* is the local part of the global general matrix *B*. This identifies the **first element** of the local array *B*. This subroutine computes the location of the first element of the local subarray used, based on *ib*, *jb*, *desc\_b*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading  $LOCp(ib+m-1)$  by  $LOCq(jb+n-1)$  part of the local array *B* must contain the local pieces of the leading  $ib+m-1$  by  $jb+n-1$  part of the global matrix.

Scope: **local**

Specified as: an  $LLD\_B$  by (at least)  $LOCq(N\_B)$  array, containing numbers of the data type indicated in Table 60 on page 256. Details about the block-cyclic data distribution of global matrix *B* are stored in *desc\_b*.

*ib* is the row index of the global matrix *B*, identifying the first row of the submatrix *B*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ib \leq M\_B$  and  $ib+m-1 \leq M\_B$ .

*jb* is the column index of the global matrix *B*, identifying the first column of the submatrix *B*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq jb \leq N\_B$  and  $jb+n-1 \leq N\_B$ .

*desc\_b* is the array descriptor for global matrix *B*, described in the following table:

<i>desc_b</i>	Name	Description	Limits	Scope
1	DTYPE_B	Descriptor type	DTYPE_B=1	Global
2	CTXT_B	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_B	Number of rows in the global matrix	If $m = 0$ or $n = 0$ : $M\_B \geq 0$ Otherwise: $M\_B \geq 1$	Global
4	N_B	Number of columns in the global matrix	If $m = 0$ or $n = 0$ : $N\_B \geq 0$ Otherwise: $N\_B \geq 1$	Global
5	MB_B	Row block size	$MB\_B \geq 1$	Global
6	NB_B	Column block size	$NB\_B \geq 1$	Global
7	RSRC_B	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_B < p$	Global
8	CSRC_B	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_B < q$	Global
9	LLD_B	The leading dimension of the local array	$LLD\_B \geq \max(1, LOCp(M\_B))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

*beta* is the scalar  $\beta$ .

Scope: **global**

Specified as: a number of the data type indicated in Table 60 on page 256.

*c* is the local part of the global general matrix *C*. This identifies the **first element** of the local array *C*. This subroutine computes the location of the first element of the local subarray used, based on *ic*, *jc*, *desc\_c*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading  $LOCp(ic+m-1)$  by  $LOCq(jc+n-1)$  part of the local array *C* must contain the local pieces of the leading  $ic+m-1$  by  $jc+n-1$  part of the global matrix.

When  $\beta$  is zero, *C* need not be set on input.

Scope: **local**

Specified as: an  $LLD\_C$  by (at least)  $LOCq(N\_C)$  array, containing numbers of the data type indicated in Table 60 on page 256. Details about the block-cyclic data distribution of global matrix *C* are stored in *desc\_c*.

## PDSYMM, PZSYMM, and PZHEMM

*ic* is the row index of the global matrix *C*, identifying the first row of the submatrix *C*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ic \leq M\_C$  and  $ic+m-1 \leq M\_C$ .

*jc* is the column index of the global matrix *C*, identifying the first column of the submatrix *C*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq jc \leq N\_C$  and  $jc+n-1 \leq N\_C$ .

*desc\_c* is the array descriptor for global matrix *C*, described in the following table:

<i>desc_c</i>	Name	Description	Limits	Scope
1	DTYPE_C	Descriptor type	DTYPE_C=1	Global
2	CTXT_C	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_C	Number of rows in the global matrix	If $m = 0$ or $n = 0$ : $M\_C \geq 0$ Otherwise: $M\_C \geq 1$	Global
4	N_C	Number of columns in the global matrix	If $m = 0$ or $n = 0$ : $N\_C \geq 0$ Otherwise: $N\_C \geq 1$	Global
5	MB_C	Row block size	$MB\_C \geq 1$	Global
6	NB_C	Column block size	$NB\_C \geq 1$	Global
7	RSRC_C	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_C < p$	Global
8	CSRC_C	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_C < q$	Global
9	LLD_C	The leading dimension of the local array	$LLD\_C \geq \max(1, LOCp(M\_C))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

### On Return

*c* is the updated local part of the global matrix *C*, containing the results of the computation.

Scope: **local**

Returned as: an LLD\_C by (at least) LOCq(N\_C) array, containing numbers of the data type indicated in Table 60 on page 256.

## Notes and Coding Rules

1. These subroutines accept lowercase letters for the *side* and *uplo* arguments.
2. The matrices must have no common elements; otherwise, results are unpredictable.

3. The imaginary parts of the diagonal elements of a complex Hermitian matrix  $A$  are assumed to be zero, so you do not have to set these values.
4. The NUMROC utility subroutine can be used to determine the values of  $\text{LOCp}(M\_)$  and  $\text{LOCq}(N\_)$  used in the argument descriptions above. For details, see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28 and “NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process” on page 868.
5. For suggested block sizes, see “Coding Tips for Optimizing Parallel Performance” on page 77.
6. The following values must be equal:  $\text{CTXT\_A} = \text{CTXT\_B} = \text{CTXT\_C}$ .
7. If  $\text{side} = 'L'$ :
  - In the process grid, the process row containing the first row of the submatrix  $A$  must also contain the first row of the submatrices  $B$  and  $C$ ; that is:
    - $\text{iarow} = \text{ibrow}$
    - $\text{iarow} = \text{icrow}$
    - where:
      - $\text{iarow} = \text{mod}(\text{mod}(\text{mod}(\text{ia}-1, \text{MB\_A}) + \text{RSRC\_A}), p)$
      - $\text{ibrow} = \text{mod}(\text{mod}(\text{mod}(\text{ib}-1, \text{MB\_B}) + \text{RSRC\_B}), p)$
      - $\text{icrow} = \text{mod}(\text{mod}(\text{mod}(\text{ic}-1, \text{MB\_C}) + \text{RSRC\_C}), p)$
  - If looping is required—that is, **either** of the following is true:
    - $n + \text{mod}(\text{jb}-1, \text{NB\_B}) > \text{NB\_B}$
    - $n + \text{mod}(\text{jc}-1, \text{NB\_C}) > \text{NB\_C}$
 then:
    - The following block sizes must be equal:  $\text{NB\_B} = \text{NB\_C}$ .
    - The block column offset of  $B$  must be equal to the block column offset of  $C$ ; that is,  $\text{mod}(\text{jb}-1, \text{NB\_B}) = \text{mod}(\text{jc}-1, \text{NB\_C})$ .
8. If  $\text{side} = 'R'$ :
  - In the process grid, the process column containing the first column of the submatrix  $A$  must also contain the first column of the submatrices  $B$  and  $C$ ; that is:
    - $\text{iacol} = \text{ibcol}$
    - $\text{iacol} = \text{iccol}$
    - where:
      - $\text{iacol} = \text{mod}(\text{mod}(\text{mod}(\text{ja}-1, \text{NB\_A}) + \text{CSRC\_A}), q)$
      - $\text{ibcol} = \text{mod}(\text{mod}(\text{mod}(\text{jb}-1, \text{NB\_B}) + \text{CSRC\_B}), q)$
      - $\text{iccol} = \text{mod}(\text{mod}(\text{mod}(\text{jc}-1, \text{NB\_C}) + \text{CSRC\_C}), q)$
  - If looping is required—that is, **either** of the following is true:
    - $m + \text{mod}(\text{ib}-1, \text{MB\_B}) > \text{MB\_B}$
    - $m + \text{mod}(\text{ic}-1, \text{MB\_C}) > \text{MB\_C}$
 then:
    - The following block sizes must be equal:  $\text{MB\_B} = \text{MB\_C}$ .
    - The block row offset of  $B$  must be equal to the block row offset of  $C$ ; that is,  $\text{mod}(\text{ib}-1, \text{MB\_B}) = \text{mod}(\text{ic}-1, \text{MB\_C})$
9. If all the following are true:
  - $A$  is contained within a single block, that is:
    - $\text{numa} + \text{mod}(\text{ia}-1, \text{MB\_A}) \leq \text{MB\_A}$
    - $\text{numa} + \text{mod}(\text{ja}-1, \text{NB\_A}) \leq \text{NB\_A}$
    - where:
      - If  $\text{side} = 'L'$ ,  $\text{numa} = m$
      - If  $\text{side} = 'R'$ ,  $\text{numa} = n$

- If *side* = 'L', then (in the process grid) the process column containing the first column of the submatrix *B* must also contain the first column of the submatrix *C*, that is,  $ibcol = iccol$ , where:
  - $ibcol = \text{mod}(((jb-1)/NB\_B)+CSRC\_B), q)$
  - $iccol = \text{mod}(((jc-1)/NB\_C)+CSRC\_C), q)$
- If *side* = 'R', then (in the process grid) the process row containing the first row of the submatrix *B* must also contain the first row of the submatrix *C*; that is,  $ibrow = icrow$ , where:
  - $ibrow = \text{mod}(((ib-1)/MB\_B)+RSRC\_B), p)$
  - $icrow = \text{mod}(((ic-1)/MB\_C)+RSRC\_C), p)$

then you must follow these rules:

- If *side* = 'L', then *B* and *C* must be block row matrices; that is, if  $p > 1$ :
  - $m + \text{mod}(ib-1, MB\_B) \leq MB\_B$
  - $m + \text{mod}(ic-1, MB\_C) \leq MB\_C$
- If *side* = 'R', then *B* and *C* must be block column matrices; that is, if  $q > 1$ :
  - $n + \text{mod}(jb-1, NB\_B) \leq NB\_B$
  - $n + \text{mod}(jc-1, NB\_C) \leq NB\_C$

10. If the following is true:

- *A* is **not** contained within a single block.

**or** if all the following are true:

- *A* is contained within a single block.
- If *side* = 'L', then (in the process grid) the process column containing the first column of the submatrix *B* does not contain the first column of the submatrix *C*, that is,  $ibcol \neq iccol$ , where:
  - $ibcol = \text{mod}(((jb-1)/NB\_B)+CSRC\_B), q)$
  - $iccol = \text{mod}(((jc-1)/NB\_C)+CSRC\_C), q)$
- If *side* = 'R', then (in the process grid) the process row containing the first row of the submatrix *B* does not contain the first row of the submatrix *C*; that is,  $ibrow \neq icrow$ , where:
  - $ibrow = \text{mod}(((ib-1)/MB\_B)+RSRC\_B), p)$
  - $icrow = \text{mod}(((ic-1)/MB\_C)+RSRC\_C), p)$

then you must follow these rules:

- The global matrix *A* must be distributed using a square block-cyclic distribution; that is,  $MB\_A = NB\_A$ .
- The global matrix *A* must be aligned on a block boundary, that is:
  - $ia-1$  must be a multiple of  $MB\_A$ .
  - $ja-1$  must be a multiple of  $NB\_A$ .
- If *side* = 'L':
  - The following block sizes must be equal:  $MB\_B = MB\_C = NB\_A$ .
  - The global matrices *B* and *C* must be aligned on a block row boundary, that is:
    - $ib-1$  must be a multiple of  $MB\_B$ .
    - $ic-1$  must be a multiple of  $MB\_C$ .
- If *side* = 'R':
  - The following block sizes must be equal:  $NB\_B = NB\_C = MB\_A$ .
  - The global matrices *B* and *C* must be aligned on a block column boundary, that is:
    - $jb-1$  must be a multiple of  $NB\_B$ .
    - $jc-1$  must be a multiple of  $NB\_C$ .



## Error Conditions

### Computational Errors

None

### Resource Errors

Unable to allocate work space

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. DTYPE\_A is invalid.
2. DTYPE\_B is invalid.
3. DTYPE\_C is invalid.

#### Stage 2:

1. CTXT\_A is invalid.

#### Stage 3:

1. This subroutine was called from outside the process grid.

#### Stage 4:

1.  $side \neq 'L'$  or  $'R'$
2.  $uplo \neq 'U'$  or  $'L'$
3.  $m < 0$
4.  $n < 0$
5.  $M\_A < 0$  and  $m = 0$  and  $side = 'L'$ ;  $M\_A < 0$  and  $n = 0$  and  $side = 'R'$ ;  $M\_A < 1$  otherwise
6.  $N\_A < 0$  and  $m = 0$  and  $side = 'L'$ ;  $N\_A < 0$  and  $n = 0$  and  $side = 'R'$ ;  $N\_A < 1$  otherwise
7.  $MB\_A < 1$
8.  $NB\_A < 1$
9.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
10.  $CSRC\_A < 0$  or  $CSRC\_A \geq q$
11.  $ia < 1$
12.  $ja < 1$
13.  $M\_B < 0$  and  $(m = 0 \text{ or } n = 0)$ ;  $M\_B < 1$  otherwise
14.  $N\_B < 0$  and  $(m = 0 \text{ or } n = 0)$ ;  $N\_B < 1$  otherwise
15.  $MB\_B < 1$
16.  $NB\_B < 1$
17.  $RSRC\_B < 0$  or  $RSRC\_B \geq p$
18.  $CSRC\_B < 0$  or  $CSRC\_B \geq q$
19.  $ib < 1$
20.  $jb < 1$
21.  $M\_C < 0$  and  $(m = 0 \text{ or } n = 0)$ ;  $M\_C < 1$  otherwise
22.  $N\_C < 0$  and  $(m = 0 \text{ or } n = 0)$ ;  $N\_C < 1$  otherwise
23.  $MB\_C < 1$
24.  $NB\_C < 1$
25.  $RSRC\_C < 0$  or  $RSRC\_C \geq p$
26.  $CSRC\_C < 0$  or  $CSRC\_C \geq q$
27.  $ic < 1$
28.  $jc < 1$
29.  $CTXT\_A \neq CTXT\_B$
30.  $CTXT\_A \neq CTXT\_C$

**Stage 5:** If  $(m \neq 0 \text{ or } side \neq 'L')$  and  $(n \neq 0 \text{ or } side \neq 'R')$ :

## PDSYMM, PZSYMM, and PZHEMM

1.  $ia > M\_A$
2.  $ja > N\_A$
3.  $ia+numa-1 > M\_A$
4.  $ja+numa-1 > N\_A$

where  $numa = m$  if  $side = 'L'$  and  $numa = n$  if  $side = 'R'$ .

If  $m \neq 0$  and  $n \neq 0$ :

1.  $ib > M\_B$
2.  $jb > N\_B$
3.  $ib+m-1 > M\_B$
4.  $jb+n-1 > N\_B$
5.  $ic > M\_C$
6.  $jc > N\_C$
7.  $ic+m-1 > M\_C$
8.  $jc+n-1 > N\_C$

**Stage 6:** If  $A$  is contained within a single block, that is:

- $numa + \text{mod}(ia-1, MB\_A) \leq MB\_A$
- $numa + \text{mod}(ja-1, NB\_A) \leq NB\_A$
- where:
  - If  $side = 'L'$ ,  $numa = m$
  - If  $side = 'R'$ ,  $numa = n$

and:

- If  $side = 'L'$ , then (in the process grid) the process column containing the first column of the submatrix  $B$  must also contain the first column of the submatrix  $C$ , that is,  $ibcol = iccol$ , where:
  - $ibcol = \text{mod}((((jb-1)/NB\_B)+CSRC\_B), q)$
  - $iccol = \text{mod}((((jc-1)/NB\_C)+CSRC\_C), q)$
- If  $side = 'R'$ , then (in the process grid) the process row containing the first row of the submatrix  $B$  must also contain the first row of the submatrix  $C$ ; that is,  $ibrow = icrow$ , where:
  - $ibrow = \text{mod}((((ib-1)/MB\_B)+RSRC\_B), p)$
  - $icrow = \text{mod}((((ic-1)/MB\_C)+RSRC\_C), p)$

then:

- If  $side = 'L'$ :
  1.  $p > 1$  and  $m + \text{mod}(ib-1, MB\_B) > MB\_B$
  2.  $p > 1$  and  $m + \text{mod}(ic-1, MB\_C) > MB\_C$
- If  $side = 'R'$ :
  1.  $q > 1$  and  $n + \text{mod}(jb-1, NB\_B) > NB\_B$
  2.  $q > 1$  and  $n + \text{mod}(jc-1, NB\_C) > NB\_C$

If  $A$  is **not** contained within a single block, or if  $A$  is contained within a single block and:

- If  $side = 'L'$ , then (in the process grid) the process column containing the first column of the submatrix  $B$  does not contain the first column of the submatrix  $C$ , that is,  $ibcol \neq iccol$ , where:
  - $ibcol = \text{mod}((((jb-1)/NB\_B)+CSRC\_B), q)$
  - $iccol = \text{mod}((((jc-1)/NB\_C)+CSRC\_C), q)$
- If  $side = 'R'$ , then (in the process grid) the process row containing the first row of the submatrix  $B$  does not contain the first row of the submatrix  $C$ ; that is,  $ibrow \neq icrow$ , where:
  - $ibrow = \text{mod}((((ib-1)/MB\_B)+RSRC\_B), p)$
  - $icrow = \text{mod}((((ic-1)/MB\_C)+RSRC\_C), p)$

then:

1.  $MB\_A \neq NB\_A$
2.  $\text{mod}(ia-1, MB\_A) \neq 0$
3.  $\text{mod}(ja-1, NB\_A) \neq 0$
- If  $side = 'L'$ :
4.  $MB\_B \neq NB\_A$
5.  $MB\_C \neq NB\_A$
6.  $\text{mod}(ib-1, MB\_B) \neq 0$
7.  $\text{mod}(ic-1, MB\_C) \neq 0$
- If  $side = 'R'$ :
8.  $NB\_B \neq MB\_A$
9.  $NB\_C \neq MB\_A$
10.  $\text{mod}(jb-1, NB\_B) \neq 0$
11.  $\text{mod}(jc-1, NB\_C) \neq 0$

In all cases:

1.  $LLD\_A < \max(1, \text{LOCp}(M\_A))$
2.  $LLD\_B < \max(1, \text{LOCp}(M\_B))$
3.  $LLD\_C < \max(1, \text{LOCp}(M\_C))$

If  $side = 'L'$  and looping is required—that is, **either** of the following is true:

- $n + \text{mod}(jb-1, NB\_B) > NB\_B$
- $n + \text{mod}(jc-1, NB\_C) > NB\_C$

then:

4.  $NB\_B \neq NB\_C$
5.  $\text{mod}(jb-1, NB\_B) \neq \text{mod}(jc-1, NB\_C)$ .

If  $side = 'L'$ :

6. In the process grid, the process row containing the first row of the submatrix *A* does not contain the first row of the submatrix *B*; that is,  $iarrow \neq ibrow$ , where:

- $iarrow = \text{mod}((((ia-1)/MB\_A)+RSRC\_A), p)$
- $ibrow = \text{mod}((((ib-1)/MB\_B)+RSRC\_B), p)$

7. In the process grid, the process row containing the first row of the submatrix *A* does not contain the first row of the submatrix *C*; that is,  $iarrow \neq icrow$ , where:

- $iarrow = \text{mod}((((ia-1)/MB\_A)+RSRC\_A), p)$
- $icrow = \text{mod}((((ic-1)/MB\_C)+RSRC\_C), p)$

If  $side = 'R'$  and looping is required—that is, **either** of the following is true:

- $m + \text{mod}(ib-1, MB\_B) > MB\_B$
- $m + \text{mod}(ic-1, MB\_C) > MB\_C$

then:

8.  $MB\_B \neq MB\_C$
9.  $\text{mod}(ib-1, MB\_B) \neq \text{mod}(ic-1, MB\_C)$ .

If  $side = 'R'$ :

10. In the process grid, the process column containing the first column of the submatrix *A* does not contain the first column of the submatrix *B*; that is,  $iacol \neq ibcol$ , where:

- $iacol = \text{mod}((((ja-1)/NB\_A)+CSRC\_A), q)$
- $ibcol = \text{mod}((((jb-1)/NB\_B)+CSRC\_B), q)$

11. In the process grid, the process column containing the first column of the submatrix *A* does not contain the first column of the submatrix *C*; that is,  $iacol \neq iccol$ , where:

- $iacol = \text{mod}((((ja-1)/NB\_A)+CSRC\_A), q)$
- $iccol = \text{mod}((((jc-1)/NB\_C)+CSRC\_C), q)$

## Examples

### Example 1

This example computes  $C = \beta C + \alpha BA$  using a  $2 \times 2$  process grid.

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      SIDE  UPLO  M   N   ALPHA   A  IA  JA  DESC_A  B  IB  JB
      |    |    |   |   |       |  |  |  |       |  |  |
CALL PDSYMM( 'R' , 'U' , 16 , 8 , 1.0D0 , A , 1 , 1 , DESC_A , B , 1 , 1 ,

      DESC_B  BETA  C  IC  JC  DESC_C
      |      |    |  |  |  |
      DESC_B , 0.0D0 , C , 1 , 1 , DESC_C )
```

	Desc_A	Desc_B	Desc_C
DTYPE_	1	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	8	16	16
N_	8	8	8
MB_	2	4	4
NB_	2	2	2
RSRC_	0	0	0
CSRC_	0	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>	See below <sup>2</sup>

#### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:

```
LLD_A = MAX(1, NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_B = MAX(1, NUMROC(M_B, MB_B, MYROW, RSRC_B, NPROW))
LLD_C = MAX(1, NUMROC(M_C, MB_C, MYROW, RSRC_C, NPROW))
```

In this example, LLD\_A = 4 on all processes, and LLD\_B = LLD\_C = 8 on all processes.

Global real symmetric matrix *A* of order 8 with block size  $2 \times 2$ :

B,D	0	1	2	3
0	0.0 -1.0 . 1.0	-1.0 0.0 0.0 1.0	0.0 0.0 0.0 1.0	0.0 0.0 0.0 1.0
1	. . . .	-1.0 -1.0 . -1.0	0.0 0.0 1.0 1.0	1.0 0.0 0.0 1.0
2	. . . .	. . . .	-1.0 0.0 . 1.0	0.0 0.0 0.0 0.0

$$3 \left[ \begin{array}{cc|cc|cc|cc} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & 0.0 & 0.0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & . & 0.0 \end{array} \right]$$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>
3		

Local arrays for *A*:

p,q	0				1			
0	0.0	-1.0	0.0	0.0	-1.0	0.0	0.0	0.0
	.	1.0	0.0	1.0	0.0	1.0	0.0	1.0
	.	.	-1.0	0.0	.	.	0.0	0.0
	.	.	.	1.0	.	.	0.0	0.0
1	.	.	0.0	0.0	-1.0	-1.0	1.0	0.0
	.	.	1.0	1.0	.	-1.0	0.0	1.0
	.	.	.	.	.	.	0.0	0.0
	.	.	.	.	.	.	.	0.0

Global general  $16 \times 8$  matrix *B* with block size  $4 \times 2$ :

B,D	0	1	2	3
0	-1.0 0.0 -1.0 -1.0 1.0 1.0 0.0 -1.0	1.0 -1.0 1.0 0.0 -1.0 0.0 0.0 0.0	1.0 1.0 1.0 -1.0 -1.0 0.0 0.0 0.0	-1.0 -1.0 -1.0 1.0 1.0 0.0 0.0 -1.0
	<hr/>			
	0.0 1.0 0.0 0.0 1.0 1.0 0.0 0.0	0.0 1.0 1.0 0.0 0.0 0.0 -1.0 0.0	0.0 1.0 -1.0 -1.0 1.0 1.0 0.0 1.0	1.0 0.0 0.0 0.0 0.0 -1.0 0.0 1.0
	<hr/>			
1	0.0 0.0 -1.0 -1.0 0.0 0.0 0.0 0.0	0.0 -1.0 1.0 0.0 0.0 1.0 1.0 1.0	1.0 1.0 0.0 -1.0 1.0 0.0 0.0 -1.0	0.0 1.0 0.0 1.0 0.0 0.0 0.0 0.0
	<hr/>			
	1.0 1.0 0.0 0.0 0.0 1.0 -1.0 0.0	-1.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0	-1.0 -1.0 1.0 0.0 0.0 0.0 0.0 1.0	1.0 1.0 0.0 -1.0 0.0 0.0 0.0 0.0
	<hr/>			
2	1.0 1.0 0.0 0.0 0.0 1.0 -1.0 0.0	-1.0 0.0 0.0 0.0 0.0 0.0 -1.0 0.0	-1.0 -1.0 1.0 0.0 0.0 0.0 0.0 1.0	1.0 1.0 0.0 -1.0 0.0 0.0 1.0 0.0
	<hr/>			
	0.0 0.0 -1.0 -1.0 0.0 0.0 0.0 0.0	0.0 -1.0 1.0 0.0 0.0 1.0 1.0 1.0	1.0 1.0 0.0 -1.0 1.0 0.0 0.0 -1.0	0.0 1.0 0.0 1.0 0.0 0.0 0.0 0.0
	<hr/>			
3	0.0 0.0 -1.0 -1.0 0.0 0.0 -1.0 0.0	0.0 -1.0 1.0 0.0 0.0 0.0 -1.0 0.0	1.0 1.0 0.0 -1.0 0.0 0.0 0.0 1.0	0.0 1.0 0.0 1.0 0.0 0.0 1.0 0.0
	<hr/>			
	1.0 1.0 0.0 0.0 0.0 1.0 -1.0 0.0	-1.0 0.0 0.0 0.0 0.0 0.0 -1.0 0.0	-1.0 -1.0 1.0 0.0 0.0 0.0 0.0 1.0	1.0 1.0 0.0 -1.0 0.0 0.0 1.0 0.0
	<hr/>			

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>
3		

Local arrays for *B*:

p,q	0	1
	-1.0 0.0 1.0 1.0	1.0 -1.0 -1.0 -1.0

## PDSYMM, PZSYMM, and PZHEMM

0	-1.0	-1.0	1.0	-1.0	1.0	0.0	-1.0	1.0
	1.0	1.0	-1.0	0.0	-1.0	0.0	1.0	0.0
	0.0	-1.0	0.0	0.0	0.0	0.0	0.0	-1.0
	0.0	0.0	1.0	1.0	0.0	-1.0	0.0	1.0
	-1.0	-1.0	0.0	-1.0	1.0	0.0	0.0	1.0
	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0
	0.0	0.0	0.0	-1.0	1.0	1.0	0.0	0.0
	-----				-----			
1	0.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0
	0.0	0.0	-1.0	-1.0	1.0	0.0	0.0	0.0
	1.0	1.0	1.0	1.0	0.0	0.0	0.0	-1.0
	0.0	0.0	0.0	1.0	-1.0	0.0	0.0	1.0
	1.0	1.0	-1.0	-1.0	-1.0	0.0	1.0	1.0
	0.0	0.0	1.0	0.0	0.0	0.0	0.0	-1.0
	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
	-1.0	0.0	0.0	1.0	-1.0	0.0	1.0	0.0

### Output:

Global general  $16 \times 8$  matrix  $C$  with block size  $4 \times 2$ :

B,D	0		1		2		3	
0	-1.0	0.0	0.0	1.0	-2.0	0.0	1.0	-1.0
	0.0	0.0	-1.0	-1.0	-1.0	-2.0	1.0	-1.0
	0.0	0.0	1.0	1.0	1.0	1.0	-1.0	1.0
	1.0	-2.0	0.0	-2.0	0.0	-1.0	0.0	-1.0
1	-1.0	3.0	0.0	1.0	1.0	3.0	0.0	2.0
	-1.0	-1.0	-1.0	-3.0	1.0	-1.0	1.0	0.0
	-1.0	0.0	-1.0	2.0	-1.0	2.0	0.0	1.0
	1.0	2.0	1.0	3.0	0.0	1.0	-1.0	0.0
2	0.0	1.0	1.0	4.0	-2.0	0.0	0.0	-1.0
	0.0	0.0	0.0	-2.0	0.0	-2.0	1.0	-1.0
	0.0	1.0	-1.0	0.0	0.0	1.0	0.0	1.0
	-1.0	0.0	-2.0	-3.0	1.0	0.0	1.0	1.0
3	0.0	0.0	1.0	1.0	1.0	0.0	-1.0	1.0
	0.0	-1.0	0.0	0.0	-1.0	0.0	0.0	0.0
	-1.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0
	1.0	2.0	3.0	2.0	0.0	1.0	-1.0	0.0

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$
3		

Local arrays for  $C$ :

p,q	0				1			
0	-1.0	0.0	-2.0	0.0	0.0	1.0	1.0	-1.0
	0.0	0.0	-1.0	-2.0	-1.0	-1.0	1.0	-1.0
	0.0	0.0	1.0	1.0	1.0	1.0	-1.0	1.0
	1.0	-2.0	0.0	-1.0	0.0	-2.0	0.0	-1.0
	0.0	1.0	-2.0	0.0	1.0	4.0	0.0	-1.0
	0.0	0.0	0.0	-2.0	0.0	-2.0	1.0	-1.0
	0.0	1.0	0.0	1.0	-1.0	0.0	0.0	1.0
	-1.0	0.0	1.0	0.0	-2.0	-3.0	1.0	1.0

$$1 \begin{vmatrix} -1.0 & 3.0 & 1.0 & 3.0 \\ -1.0 & -1.0 & 1.0 & -1.0 \\ -1.0 & 0.0 & -1.0 & 2.0 \\ 1.0 & 2.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & -1.0 & -1.0 & 0.0 \\ -1.0 & 1.0 & 0.0 & 1.0 \\ 1.0 & 2.0 & 0.0 & 1.0 \end{vmatrix} \begin{vmatrix} 0.0 & 1.0 & 0.0 & 2.0 \\ -1.0 & -3.0 & 1.0 & 0.0 \\ -1.0 & 2.0 & 0.0 & 1.0 \\ 1.0 & 3.0 & -1.0 & 0.0 \\ 1.0 & 1.0 & -1.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 1.0 \\ 3.0 & 2.0 & -1.0 & 0.0 \end{vmatrix}$$

## Example 2

This example computes  $C = \beta C + \alpha BA$  using a  $2 \times 2$  process grid.

### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      SIDE  UPLO  M  N      ALPHA      A  IA  JA  DESC_A  B  IB  JB
      |      |      |      |      |      |  |  |  |      |  |  |
CALL PZSYMM( 'R' , 'U' , 16 , 8 ,  ALPHA      , A , 1 , 1 , DESC_A , B , 1 , 1 ,

      DESC_B  BETA      C  IC  JC  DESC_C
      |      |      |  |  |  |
      DESC_B , BETA      , C , 1 , 1 , DESC_C )

ALPHA = (1.0, 2.0)
BETA  = (0.0, 0.0)
```

	Desc_A	Desc_B	Desc_C
DTYPE_	1	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	8	16	16
N_	8	8	8
MB_	2	4	4
NB_	2	2	2
RSRC_	0	0	0
CSRC_	0	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>	See below <sup>2</sup>

### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:

```
LLD_A = MAX(1, NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_B = MAX(1, NUMROC(M_B, MB_B, MYROW, RSRC_B, NPROW))
LLD_C = MAX(1, NUMROC(M_C, MB_C, MYROW, RSRC_C, NPROW))
```

In this example, LLD\_A = 4 on all processes, and LLD\_B = LLD\_C = 8 on all processes.

Global complex symmetric matrix *A* of order 8 with block size  $2 \times 2$ :

## PDSYMM, PZSYMM, and PZHEMM

B,D	0	1	2	3
0	$\begin{pmatrix} (0.0, 1.0) & (-1.0, 0.0) \\ . & (1.0, 2.0) \end{pmatrix}$	$\begin{pmatrix} (-1.0, 0.0) & (0.0, 1.0) \\ (0.0, 1.0) & (1.0, 2.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) & (0.0, 1.0) \\ (0.0, 1.0) & (1.0, 2.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) & (0.0, 1.0) \\ (0.0, 1.0) & (1.0, 2.0) \end{pmatrix}$
1	$\begin{pmatrix} . & . \\ . & . \end{pmatrix}$	$\begin{pmatrix} (-1.0, 0.0) & (-1.0, 0.0) \\ . & (-1.0, 0.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) & (0.0, 1.0) \\ (1.0, 2.0) & (1.0, 2.0) \end{pmatrix}$	$\begin{pmatrix} (1.0, 2.0) & (0.0, 1.0) \\ (0.0, 1.0) & (1.0, 2.0) \end{pmatrix}$
2	$\begin{pmatrix} . & . \\ . & . \end{pmatrix}$	$\begin{pmatrix} . & . \\ . & . \end{pmatrix}$	$\begin{pmatrix} (-1.0, 0.0) & (0.0, 1.0) \\ . & (1.0, 2.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) & (0.0, 1.0) \\ (0.0, 1.0) & (0.0, 1.0) \end{pmatrix}$
3	$\begin{pmatrix} . & . \\ . & . \end{pmatrix}$	$\begin{pmatrix} . & . \\ . & . \end{pmatrix}$	$\begin{pmatrix} . & . \\ . & . \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) & (0.0, 1.0) \\ . & (0.0, 1.0) \end{pmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$
3		

Local arrays for A:

p,q	0	1
0	$\begin{pmatrix} (0.0, 1.0) & (-1.0, 0.0) & (0.0, 1.0) & (0.0, 1.0) \\ . & (1.0, 2.0) & (0.0, 1.0) & (1.0, 2.0) \\ . & . & (-1.0, 0.0) & (0.0, 1.0) \\ . & . & . & (1.0, 2.0) \end{pmatrix}$	$\begin{pmatrix} (-1.0, 0.0) & (0.0, 1.0) & (0.0, 1.0) & (0.0, 1.0) \\ (0.0, 1.0) & (1.0, 2.0) & (0.0, 1.0) & (1.0, 2.0) \\ . & . & (0.0, 1.0) & (0.0, 1.0) \\ . & . & (0.0, 1.0) & (0.0, 1.0) \end{pmatrix}$
1	$\begin{pmatrix} . & . & (0.0, 1.0) & (0.0, 1.0) \\ . & . & (1.0, 2.0) & (1.0, 2.0) \\ . & . & . & . \\ . & . & . & . \end{pmatrix}$	$\begin{pmatrix} (-1.0, 0.0) & (-1.0, 0.0) & (1.0, 2.0) & (0.0, 1.0) \\ . & (-1.0, 0.0) & (0.0, 1.0) & (1.0, 2.0) \\ . & . & (0.0, 1.0) & (0.0, 1.0) \\ . & . & . & (0.0, 1.0) \end{pmatrix}$

Global general  $16 \times 8$  matrix  $B$  with block size  $4 \times 2$ :

B,D	0	1	2	3
0	$\begin{pmatrix} (-1.0, -3.0) & (0.0, -2.0) \\ (-1.0, -3.0) & (-1.0, -3.0) \\ (1.0, -1.0) & (1.0, -1.0) \\ (0.0, -2.0) & (-1.0, -3.0) \end{pmatrix}$	$\begin{pmatrix} (1.0, -1.0) & (-1.0, -3.0) \\ (1.0, -1.0) & (0.0, -2.0) \\ (-1.0, -3.0) & (0.0, -2.0) \\ (0.0, -2.0) & (0.0, -2.0) \end{pmatrix}$	$\begin{pmatrix} (1.0, -1.0) & (1.0, -1.0) \\ (1.0, -1.0) & (-1.0, -3.0) \\ (-1.0, -3.0) & (0.0, -2.0) \\ (0.0, -2.0) & (0.0, -2.0) \end{pmatrix}$	$\begin{pmatrix} (-1.0, -3.0) & (-1.0, -3.0) \\ (-1.0, -3.0) & (1.0, -1.0) \\ (1.0, -1.0) & (0.0, -2.0) \\ (0.0, -2.0) & (-1.0, -3.0) \end{pmatrix}$
1	$\begin{pmatrix} (0.0, -2.0) & (1.0, -1.0) \\ (0.0, -2.0) & (0.0, -2.0) \\ (1.0, -1.0) & (1.0, -1.0) \\ (0.0, -2.0) & (0.0, -2.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, -2.0) & (1.0, -1.0) \\ (1.0, -1.0) & (0.0, -2.0) \\ (0.0, -2.0) & (0.0, -2.0) \\ (-1.0, -3.0) & (0.0, -2.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, -2.0) & (1.0, -1.0) \\ (-1.0, -3.0) & (-1.0, -3.0) \\ (1.0, -1.0) & (1.0, -1.0) \\ (0.0, -2.0) & (1.0, -1.0) \end{pmatrix}$	$\begin{pmatrix} (1.0, -1.0) & (0.0, -2.0) \\ (0.0, -2.0) & (0.0, -2.0) \\ (0.0, -2.0) & (-1.0, -3.0) \\ (0.0, -2.0) & (1.0, -1.0) \end{pmatrix}$
2	$\begin{pmatrix} (0.0, -2.0) & (0.0, -2.0) \\ (-1.0, -3.0) & (-1.0, -3.0) \\ (0.0, -2.0) & (0.0, -2.0) \\ (0.0, -2.0) & (0.0, -2.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, -2.0) & (-1.0, -3.0) \\ (1.0, -1.0) & (0.0, -2.0) \\ (0.0, -2.0) & (1.0, -1.0) \\ (1.0, -1.0) & (1.0, -1.0) \end{pmatrix}$	$\begin{pmatrix} (1.0, -1.0) & (1.0, -1.0) \\ (0.0, -2.0) & (-1.0, -3.0) \\ (1.0, -1.0) & (0.0, -2.0) \\ (0.0, -2.0) & (-1.0, -3.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, -2.0) & (1.0, -1.0) \\ (0.0, -2.0) & (1.0, -1.0) \\ (0.0, -2.0) & (0.0, -2.0) \\ (0.0, -2.0) & (0.0, -2.0) \end{pmatrix}$
3	$\begin{pmatrix} (1.0, -1.0) & (1.0, -1.0) \\ (0.0, -2.0) & (0.0, -2.0) \\ (0.0, -2.0) & (1.0, -1.0) \\ (-1.0, -3.0) & (0.0, -2.0) \end{pmatrix}$	$\begin{pmatrix} (-1.0, -3.0) & (0.0, -2.0) \\ (0.0, -2.0) & (0.0, -2.0) \\ (0.0, -2.0) & (0.0, -2.0) \\ (-1.0, -3.0) & (0.0, -2.0) \end{pmatrix}$	$\begin{pmatrix} (-1.0, -3.0) & (-1.0, -3.0) \\ (1.0, -1.0) & (0.0, -2.0) \\ (0.0, -2.0) & (0.0, -2.0) \\ (0.0, -2.0) & (1.0, -1.0) \end{pmatrix}$	$\begin{pmatrix} (1.0, -1.0) & (1.0, -1.0) \\ (0.0, -2.0) & (-1.0, -3.0) \\ (0.0, -2.0) & (0.0, -2.0) \\ (1.0, -1.0) & (0.0, -2.0) \end{pmatrix}$

The following is the  $2 \times 2$  process grid:



B,D	0 2	1 3
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>
3		

Local arrays for *B*:

p,q	0	1
0	(-1.0,-3.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 1.0,-1.0) (-1.0,-3.0) (-1.0,-3.0) ( 1.0,-1.0) (-1.0,-3.0) ( 1.0,-1.0) ( 1.0,-1.0) (-1.0,-3.0) ( 0.0,-2.0) ( 0.0,-2.0) (-1.0,-3.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 1.0,-1.0) (-1.0,-3.0) (-1.0,-3.0) ( 0.0,-2.0) (-1.0,-3.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) (-1.0,-3.0)	( 1.0,-1.0) (-1.0,-3.0) (-1.0,-3.0) (-1.0,-3.0) ( 1.0,-1.0) ( 0.0,-2.0) (-1.0,-3.0) ( 1.0,-1.0) (-1.0,-3.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) (-1.0,-3.0) ( 0.0,-2.0) (-1.0,-3.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0)
1	( 0.0,-2.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) (-1.0,-3.0) (-1.0,-3.0) ( 1.0,-1.0) ( 1.0,-1.0) ( 1.0,-1.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 1.0,-1.0) ( 1.0,-1.0) (-1.0,-3.0) (-1.0,-3.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) (-1.0,-3.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 1.0,-1.0)	( 0.0,-2.0) ( 1.0,-1.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) (-1.0,-3.0) (-1.0,-3.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 1.0,-1.0) (-1.0,-3.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) (-1.0,-3.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) (-1.0,-3.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 0.0,-2.0)

Output:

Global general  $16 \times 8$  matrix *C* with block size  $4 \times 2$ :

B,D	0	1	2	3
0	(11.0,27.0) (37.0,39.0) ( 7.0,29.0) (37.0,39.0) ( 1.0,27.0) (31.0,37.0) ( 6.0,32.0) (48.0,36.0)	( 7.0,29.0) (27.0,39.0) (11.0,27.0) (35.0,35.0) (-3.0,29.0) (21.0,37.0) (10.0,30.0) (42.0,34.0)	(27.0,29.0) (37.0,39.0) (23.0,31.0) (45.0,35.0) ( 9.0,33.0) (27.0,39.0) (22.0,34.0) (44.0,38.0)	(21.0,37.0) (35.0,35.0) (21.0,37.0) (35.0,35.0) (23.0,31.0) (21.0,37.0) (28.0,36.0) (38.0,36.0)
1	(-4.0,22.0) (10.0,40.0) (11.0,27.0) (41.0,37.0) (-1.0,23.0) (25.0,35.0) (-3.0,29.0) (23.0,41.0)	(-8.0,24.0) (12.0,34.0) (11.0,27.0) (43.0,31.0) (-1.0,23.0) (11.0,37.0) (-3.0,29.0) (13.0,41.0)	( 0.0,30.0) (10.0,40.0) (15.0,35.0) (41.0,37.0) (11.0,27.0) (17.0,39.0) (13.0,31.0) (27.0,39.0)	(10.0,30.0) ( 8.0,36.0) (21.0,37.0) (31.0,37.0) (13.0,31.0) (15.0,35.0) (23.0,31.0) (25.0,35.0)
2	(-2.0,26.0) (24.0,38.0) ( 7.0,29.0) (37.0,39.0) (-2.0,26.0) (24.0,38.0) ( 5.0,25.0) (31.0,37.0)	(-6.0,28.0) ( 6.0,42.0) ( 7.0,29.0) (39.0,33.0) ( 2.0,24.0) (22.0,34.0) ( 9.0,23.0) (37.0,29.0)	(18.0,26.0) (28.0,36.0) (19.0,33.0) (45.0,35.0) (10.0,30.0) (24.0,38.0) ( 9.0,33.0) (31.0,37.0)	(16.0,32.0) (26.0,32.0) (21.0,37.0) (35.0,35.0) (16.0,32.0) (18.0,36.0) (15.0,35.0) (21.0,37.0)
3	( 1.0,27.0) (31.0,37.0) ( 4.0,28.0) (38.0,36.0) ( 5.0,25.0) (27.0,39.0) ( 0.0,30.0) (26.0,42.0)	(-3.0,29.0) (21.0,37.0) ( 4.0,28.0) (28.0,36.0) ( 1.0,27.0) (21.0,37.0) (-8.0,34.0) (20.0,40.0)	( 9.0,33.0) (31.0,37.0) (20.0,30.0) (34.0,38.0) (13.0,31.0) (27.0,39.0) (16.0,32.0) (30.0,40.0)	(23.0,31.0) (21.0,37.0) (22.0,34.0) (28.0,36.0) (19.0,33.0) (21.0,37.0) (26.0,32.0) (28.0,36.0)

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>
3		

## PDSYMM, PZSYMM, and PZHEMM

Local arrays for C:

p,q	0				1			
0	(11.0,27.0)	(37.0,39.0)	(27.0,29.0)	(37.0,39.0)	( 7.0,29.0)	(27.0,39.0)	(21.0,37.0)	(35.0,35.0)
	( 7.0,29.0)	(37.0,39.0)	(23.0,31.0)	(45.0,35.0)	(11.0,27.0)	(35.0,35.0)	(21.0,37.0)	(35.0,35.0)
	( 1.0,27.0)	(31.0,37.0)	( 9.0,33.0)	(27.0,39.0)	(-3.0,29.0)	(21.0,37.0)	(23.0,31.0)	(21.0,37.0)
	( 6.0,32.0)	(48.0,36.0)	(22.0,34.0)	(44.0,38.0)	(10.0,30.0)	(42.0,34.0)	(28.0,36.0)	(38.0,36.0)
	(-2.0,26.0)	(24.0,38.0)	(18.0,26.0)	(28.0,36.0)	(-6.0,28.0)	( 6.0,42.0)	(16.0,32.0)	(26.0,32.0)
	( 7.0,29.0)	(37.0,39.0)	(19.0,33.0)	(45.0,35.0)	( 7.0,29.0)	(39.0,33.0)	(21.0,37.0)	(35.0,35.0)
	(-2.0,26.0)	(24.0,38.0)	(10.0,30.0)	(24.0,38.0)	( 2.0,24.0)	(22.0,34.0)	(16.0,32.0)	(18.0,36.0)
	( 5.0,25.0)	(31.0,37.0)	( 9.0,33.0)	(31.0,37.0)	( 9.0,23.0)	(37.0,29.0)	(15.0,35.0)	(21.0,37.0)
1	(-4.0,22.0)	(10.0,40.0)	( 0.0,30.0)	(10.0,40.0)	(-8.0,24.0)	(12.0,34.0)	(10.0,30.0)	( 8.0,36.0)
	(11.0,27.0)	(41.0,37.0)	(15.0,35.0)	(41.0,37.0)	(11.0,27.0)	(43.0,31.0)	(21.0,37.0)	(31.0,37.0)
	(-1.0,23.0)	(25.0,35.0)	(11.0,27.0)	(17.0,39.0)	(-1.0,23.0)	(11.0,37.0)	(13.0,31.0)	(15.0,35.0)
	(-3.0,29.0)	(23.0,41.0)	(13.0,31.0)	(27.0,39.0)	(-3.0,29.0)	(13.0,41.0)	(23.0,31.0)	(25.0,35.0)
	( 1.0,27.0)	(31.0,37.0)	( 9.0,33.0)	(31.0,37.0)	(-3.0,29.0)	(21.0,37.0)	(23.0,31.0)	(21.0,37.0)
	( 4.0,28.0)	(38.0,36.0)	(20.0,30.0)	(34.0,38.0)	( 4.0,28.0)	(28.0,36.0)	(22.0,34.0)	(28.0,36.0)
	( 5.0,25.0)	(27.0,39.0)	(13.0,31.0)	(27.0,39.0)	( 1.0,27.0)	(21.0,37.0)	(19.0,33.0)	(21.0,37.0)
	( 0.0,30.0)	(26.0,42.0)	(16.0,32.0)	(30.0,40.0)	(-8.0,34.0)	(20.0,40.0)	(26.0,32.0)	(28.0,36.0)

### Example 3

This example computes  $C = \beta C + \alpha BA$  using a  $2 \times 2$  process grid.

**Note:** The imaginary parts of the diagonal elements of a complex Hermitian matrix are assumed to be zero, so you do not have to set these values.

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      SIDE  UPLO  M  N  ALPHA  A  IA  JA  DESC_A  B  IB  JB
      |    |    |  |  |      |  |  |  |      |  |  |
CALL PZHEMM( 'R' , 'U' , 16 , 8 ,  ALPHA  , A , 1 , 1 , DESC_A , B , 1 , 1 ,

      DESC_B  BETA  C  IC  JC  DESC_C
      |      |    |  |  |  |
      DESC_B , BETA , C , 1 , 1 , DESC_C )

ALPHA = (1.0, 0.0)
BETA  = (0.0, 0.0)
```

	Desc_A	Desc_B	Desc_C
DTYPE_	1	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	8	16	16
N_	8	8	8
MB_	2	4	4
NB_	2	2	2
RSRC_	0	0	0
CSRC_	0	0	0

	Desc_A	Desc_B	Desc_C
LLD_	See below <sup>2</sup>	See below <sup>2</sup>	See below <sup>2</sup>

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.

2. Each process should set the LLD\_ as follows:

LLD\_A = MAX(1, NUMROC(M\_A, MB\_A, MYROW, RSRC\_A, NPROW))

LLD\_B = MAX(1, NUMROC(M\_B, MB\_B, MYROW, RSRC\_B, NPROW))

LLD\_C = MAX(1, NUMROC(M\_C, MB\_C, MYROW, RSRC\_C, NPROW))

In this example, LLD\_A = 4 on all processes, and LLD\_B = LLD\_C = 8 on all processes.

Global Hermitian matrix *A* of order 8 with block size  $2 \times 2$ :

B,D	0	1	2	3
0	( 0.0, 0.0) (-1.0, 0.0) ( 1.0, 0.0)	(-1.0, 0.0) ( 0.0, 1.0) ( 0.0, 1.0) ( 1.0, 2.0)	( 0.0, 1.0) ( 0.0, 1.0) ( 0.0, 1.0) ( 1.0, 2.0)	( 0.0, 1.0) ( 0.0, 1.0) ( 0.0, 1.0) ( 1.0, 2.0)
1	. . . .	(-1.0, 0.0) (-1.0, 0.0) . (-1.0, 0.0)	( 0.0, 1.0) ( 0.0, 1.0) ( 1.0, 2.0) ( 1.0, 2.0)	( 1.0, 2.0) ( 0.0, 1.0) ( 0.0, 1.0) ( 1.0, 2.0)
2	. . . .	. . . .	(-1.0, 0.0) ( 0.0, 1.0) . ( 1.0, 0.0)	( 0.0, 1.0) ( 0.0, 1.0) ( 0.0, 1.0) ( 0.0, 1.0)
3	. . . .	. . . .	. . . .	( 0.0, 0.0) ( 0.0, 1.0) . ( 0.0, 0.0)

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0 2	P <sub>00</sub>	P <sub>01</sub>
1 3	P <sub>10</sub>	P <sub>11</sub>

Local arrays for *A*:

p,q	0	1
0	( 0.0, . ) (-1.0, 0.0) ( 0.0, 1.0) ( 0.0, 1.0) ( 1.0, . ) ( 0.0, 1.0) ( 1.0, 2.0) . (-1.0, . ) ( 0.0, 1.0) . ( 1.0, . )	(-1.0, 0.0) ( 0.0, 1.0) ( 0.0, 1.0) ( 0.0, 1.0) ( 0.0, 1.0) ( 1.0, 2.0) ( 0.0, 1.0) ( 1.0, 2.0) . ( 0.0, 1.0) ( 0.0, 1.0) . ( 0.0, 1.0) ( 0.0, 1.0)
1	. . ( 0.0, 1.0) ( 0.0, 1.0) ( 1.0, 2.0) ( 1.0, 2.0) . . . .	(-1.0, . ) (-1.0, 0.0) ( 1.0, 2.0) ( 0.0, 1.0) . (-1.0, . ) ( 0.0, 1.0) ( 1.0, 2.0) . ( 0.0, . ) ( 0.0, 1.0) . ( 0.0, . )

Global general  $16 \times 8$  matrix *B* with block size  $4 \times 2$ :

## PDSYMM, PZSYMM, and PZHEMM

B,D	0	1	2	3
0	(-1.0,-3.0) ( 0.0,-2.0) (-1.0,-3.0) (-1.0,-3.0) ( 1.0,-1.0) ( 1.0,-1.0) ( 0.0,-2.0) (-1.0,-3.0)	( 1.0,-1.0) (-1.0,-3.0) ( 1.0,-1.0) ( 0.0,-2.0) (-1.0,-3.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0)	( 1.0,-1.0) ( 1.0,-1.0) ( 1.0,-1.0) (-1.0,-3.0) (-1.0,-3.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0)	(-1.0,-3.0) (-1.0,-3.0) (-1.0,-3.0) ( 1.0,-1.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) (-1.0,-3.0)
1	( 0.0,-2.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0)	( 0.0,-2.0) ( 1.0,-1.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) (-1.0,-3.0) ( 0.0,-2.0)	( 0.0,-2.0) ( 1.0,-1.0) (-1.0,-3.0) (-1.0,-3.0) ( 1.0,-1.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 1.0,-1.0)	( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) (-1.0,-3.0) ( 0.0,-2.0) ( 1.0,-1.0)
2	( 0.0,-2.0) ( 0.0,-2.0) (-1.0,-3.0) (-1.0,-3.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0)	( 0.0,-2.0) (-1.0,-3.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 1.0,-1.0) ( 1.0,-1.0)	( 1.0,-1.0) ( 1.0,-1.0) ( 0.0,-2.0) (-1.0,-3.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) (-1.0,-3.0)	( 0.0,-2.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0)
3	( 1.0,-1.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 1.0,-1.0) (-1.0,-3.0) ( 0.0,-2.0)	(-1.0,-3.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) (-1.0,-3.0) ( 0.0,-2.0)	(-1.0,-3.0) (-1.0,-3.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 1.0,-1.0)	( 1.0,-1.0) ( 1.0,-1.0) ( 0.0,-2.0) (-1.0,-3.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 0.0,-2.0)

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>
3		

Local arrays for *B*:

p,q	0	1
0	(-1.0,-3.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 1.0,-1.0) (-1.0,-3.0) (-1.0,-3.0) ( 1.0,-1.0) (-1.0,-3.0) ( 1.0,-1.0) ( 1.0,-1.0) (-1.0,-3.0) ( 0.0,-2.0) ( 0.0,-2.0) (-1.0,-3.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 1.0,-1.0) (-1.0,-3.0) (-1.0,-3.0) ( 0.0,-2.0) (-1.0,-3.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) (-1.0,-3.0)	( 1.0,-1.0) (-1.0,-3.0) (-1.0,-3.0) (-1.0,-3.0) ( 1.0,-1.0) ( 0.0,-2.0) (-1.0,-3.0) ( 1.0,-1.0) (-1.0,-3.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) (-1.0,-3.0) ( 0.0,-2.0) (-1.0,-3.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0)
1	( 0.0,-2.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) (-1.0,-3.0) (-1.0,-3.0) ( 1.0,-1.0) ( 1.0,-1.0) ( 1.0,-1.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 1.0,-1.0) ( 1.0,-1.0) (-1.0,-3.0) (-1.0,-3.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) (-1.0,-3.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 1.0,-1.0)	( 0.0,-2.0) ( 1.0,-1.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) (-1.0,-3.0) (-1.0,-3.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 1.0,-1.0) (-1.0,-3.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) (-1.0,-3.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) (-1.0,-3.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 0.0,-2.0)

Output:

Global general  $16 \times 8$  matrix *C* with block size  $4 \times 2$ :

B,D	0	1	2	3
0	(-12.0,4.0) (-19.0,-5.0) (-10.0,4.0) (-17.0,-7.0) (-10.0,4.0) (-19.0,-5.0) (-10.0,6.0) (-22.0,-6.0)	(-9.0, 5.0) (-5.0,-5.0) (-9.0, 3.0) (-5.0,-9.0) (-7.0, 5.0) (-11.0,1.0) (-8.0, 4.0) (-10.0,-6.0)	( 3.0,-3.0) ( 9.0,-5.0) ( 3.0,-1.0) ( 9.0,-9.0) ( 5.0, 1.0) (11.0,-5.0) ( 4.0, 0.0) (10.0,-8.0)	(10.0, 2.0) (18.0,-6.0) (14.0,-2.0) (20.0,-8.0) (12.0,-4.0) (17.0,-1.0) (12.0,-2.0) (19.0,-7.0)
1	(-8.0, 0.0) (-10.0,-8.0) (-13.0,5.0) (-21.0,-5.0) (-10.0,2.0) (-17.0,-7.0) (-7.0, 3.0) (-13.0,-7.0)	(-6.0, 2.0) (-6.0,-4.0) (-11.0,5.0) (-15.0,-3.0) (-9.0, 3.0) (-7.0,-1.0) (-5.0, 3.0) (-1.0,-5.0)	( 4.0, 2.0) (10.0, 0.0) ( 3.0, 3.0) ( 9.0,-7.0) ( 1.0, 1.0) ( 7.0, 1.0) ( 7.0,-3.0) (13.0,-7.0)	( 9.0, 1.0) (14.0, 4.0) (13.0,-1.0) (19.0,-5.0) ( 7.0, 3.0) (14.0, 2.0) (13.0,-5.0) (18.0,-4.0)
2	(-8.0, 2.0) (-14.0,-8.0) (-10.0,4.0) (-17.0,-7.0) (-8.0, 2.0) (-14.0,-8.0) (-11.0,3.0) (-17.0,-7.0)	(-4.0, 2.0) ( 2.0,-6.0) (-7.0, 3.0) (-7.0,-9.0) (-8.0, 2.0) (-6.0,-6.0) (-11.0,3.0) (-13.0,-5.0)	( 6.0,-6.0) (12.0,-8.0) ( 5.0,-1.0) (11.0,-11.0) ( 2.0, 2.0) ( 8.0,-2.0) ( 1.0, 5.0) ( 7.0,-3.0)	(12.0,-2.0) (17.0,-5.0) (15.0,-3.0) (20.0,-8.0) (10.0, 0.0) (16.0, 0.0) (11.0, 1.0) (17.0,-1.0)
3	(-10.0,4.0) (-19.0,-5.0) (-10.0,4.0) (-20.0,-6.0) (-11.0,3.0) (-17.0,-5.0) (-7.0, 3.0) (-14.0,-6.0)	(-7.0, 5.0) (-11.0,1.0) (-8.0, 4.0) (-8.0,-4.0) (-9.0, 5.0) (-9.0,-1.0) (-2.0, 4.0) (-2.0,-6.0)	( 5.0, 1.0) (11.0,-7.0) ( 2.0, 0.0) ( 8.0,-4.0) ( 3.0, 1.0) ( 9.0,-3.0) ( 8.0,-4.0) (14.0,-8.0)	(14.0,-6.0) (18.0,-2.0) (10.0, 0.0) (17.0,-3.0) (11.0,-1.0) (17.0,-1.0) (13.0,-5.0) (18.0,-4.0)

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for C:

p,q	0	1
0	(-12.0, 4.0) (-19.0, -5.0) ( 3.0, -3.0) ( 9.0, -5.0) (-10.0, 4.0) (-17.0, -7.0) ( 3.0, -1.0) ( 9.0, -9.0) (-10.0, 4.0) (-19.0, -5.0) ( 5.0, 1.0) (11.0, -5.0) (-10.0, 6.0) (-22.0, -6.0) ( 4.0, 0.0) (10.0, -8.0) (-8.0, 2.0) (-14.0, -8.0) ( 6.0, -6.0) (12.0, -8.0) (-10.0, 4.0) (-17.0, -7.0) ( 5.0, -1.0) (11.0, -11.0) (-8.0, 2.0) (-14.0, -8.0) ( 2.0, 2.0) ( 8.0, -2.0) (-11.0, 3.0) (-17.0, -7.0) ( 1.0, 5.0) ( 7.0, -3.0)	(-9.0, 5.0) (-5.0, -5.0) (10.0, 2.0) (18.0, -6.0) (-9.0, 3.0) (-5.0, -9.0) (14.0, -2.0) (20.0, -8.0) (-7.0, 5.0) (-11.0, 1.0) (12.0, -4.0) (17.0, -1.0) (-8.0, 4.0) (-10.0, -6.0) (12.0, -2.0) (19.0, -7.0) (-4.0, 2.0) ( 2.0, -6.0) (12.0, -2.0) (17.0, -5.0) (-7.0, 3.0) (-7.0, -9.0) (15.0, -3.0) (20.0, -8.0) (-8.0, 2.0) (-6.0, -6.0) (10.0, 0.0) (16.0, 0.0) (-11.0, 3.0) (-13.0, -5.0) (11.0, 1.0) (17.0, -1.0)
1	(-8.0, 0.0) (-10.0, -8.0) ( 4.0, 2.0) (10.0, 0.0) (-13.0, 5.0) (-21.0, -5.0) ( 3.0, 3.0) ( 9.0, -7.0) (-10.0, 2.0) (-17.0, -7.0) ( 1.0, 1.0) ( 7.0, 1.0) (-7.0, 3.0) (-13.0, -7.0) ( 7.0, -3.0) (13.0, -7.0) (-10.0, 4.0) (-19.0, -5.0) ( 5.0, 1.0) (11.0, -7.0) (-10.0, 4.0) (-20.0, -6.0) ( 2.0, 0.0) ( 8.0, -4.0) (-11.0, 3.0) (-17.0, -5.0) ( 3.0, 1.0) ( 9.0, -3.0) (-7.0, 3.0) (-14.0, -6.0) ( 8.0, -4.0) (14.0, -8.0)	(-6.0, 2.0) (-6.0, -4.0) ( 9.0, 1.0) (14.0, 4.0) (-11.0, 5.0) (-15.0, -3.0) (13.0, -1.0) (19.0, -5.0) (-9.0, 3.0) (-7.0, -1.0) ( 7.0, 3.0) (14.0, 2.0) (-5.0, 3.0) (-1.0, -5.0) (13.0, -5.0) (18.0, -4.0) (-7.0, 5.0) (-11.0, 1.0) (14.0, -6.0) (18.0, -2.0) (-8.0, 4.0) (-8.0, -4.0) (10.0, 0.0) (17.0, -3.0) (-9.0, 5.0) (-9.0, -1.0) (11.0, -1.0) (17.0, -1.0) (-2.0, 4.0) (-2.0, -6.0) (13.0, -5.0) (18.0, -4.0)

## PDTRMM and PZTRMM — Triangular Matrix-Matrix Product

### Purpose

PDTRMM computes one of the following matrix-matrix products:

- |                                |                                |
|--------------------------------|--------------------------------|
| 1. $B \leftarrow \alpha AB$    | 3. $B \leftarrow \alpha BA$    |
| 2. $B \leftarrow \alpha A^T B$ | 4. $B \leftarrow \alpha B A^T$ |

PZTRMM computes one of the following matrix-matrix products:

- |                                |                                |                                |
|--------------------------------|--------------------------------|--------------------------------|
| 1. $B \leftarrow \alpha AB$    | 3. $B \leftarrow \alpha BA$    | 5. $B \leftarrow \alpha A^H B$ |
| 2. $B \leftarrow \alpha A^T B$ | 4. $B \leftarrow \alpha B A^T$ | 6. $B \leftarrow \alpha B A^H$ |

where, in the formulas above:

- $A$  represents the global triangular submatrix:
  - For  $side = 'L'$ , it is  $A_{ia:ia+m-1, ja:ja+m-1}$ .
  - For  $side = 'R'$ , it is  $A_{ia:ia+n-1, ja:ja+n-1}$ .
- $B$  represents the global general submatrix  $B_{ib:ib+m-1, jb:jb+n-1}$ .
- $\alpha$  is a scalar.

**Note:** No data should be moved to form  $A^T$  or  $A^H$ ; that is, the matrix  $A$  should always be stored in its untransposed form.

If  $m = 0$  or  $n = 0$ , no computation is performed, and the subroutine returns after doing some parameter checking.

See references [15] and [16].

Table 61. Data Types

$\alpha, A, B$	Subprogram
Long-precision real	PDTRMM
Long-precision complex	PZTRMM

### Syntax

<b>Fortran</b>	CALL PDTRMM   PZTRMM ( <i>side</i> , <i>uplo</i> , <i>transa</i> , <i>diag</i> , <i>m</i> , <i>n</i> , <i>alpha</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>b</i> , <i>ib</i> , <i>jb</i> , <i>desc_b</i> )
<b>C and C++</b>	pdtrmm   pztrmm ( <i>side</i> , <i>uplo</i> , <i>transa</i> , <i>diag</i> , <i>m</i> , <i>n</i> , <i>alpha</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>b</i> , <i>ib</i> , <i>jb</i> , <i>desc_b</i> );

### On Entry

*side* indicates whether  $A$  is located to the left or right of  $B$  in the equation used for this computation, where:

If  $side = 'L'$ ,  $A$  is to the left of  $B$ .

If  $side = 'R'$ ,  $A$  is to the right of  $B$ .

Scope: **global**

Specified as: a single character;  $side = 'L'$  or  $'R'$ .

*uplo* indicates whether the upper or lower triangular part of the global triangular submatrix  $A$  is referenced, where:

If  $uplo = 'U'$ , the upper triangular part is referenced.

If *uplo* = 'L', the lower triangular part is referenced.

Scope: **global**

Specified as: a single character; *uplo* = 'U' or 'L'.

*transa* indicates the form of matrix *A* to use in the computation, where:

If *transa* = 'N', *A* is used in the computation.

If *transa* = 'T',  $A^T$  is used in the computation.

If *transa* = 'C',  $A^H$  is used in the computation.

Scope: **global**

Specified as: a single character; *transa* = 'N', 'T', or 'C'.

*diag* indicates the characteristics of the diagonal of matrix *A*, where:

If *diag* = 'U', *A* is a unit triangular matrix.

If *diag* = 'N', *A* is not a unit triangular matrix.

Scope: **global**

Specified as: a single character; *diag* = 'U' or 'N'.

*m* is the number of rows in submatrix *B*, and:

If *side* = 'L', it is the number of rows and columns in submatrix *A* used in the computation.

Scope: **global**

Specified as: a fullword integer;  $m \geq 0$ .

*n* is the number of columns in submatrix *B*, and:

If *side* = 'R', it is the number of rows and columns in submatrix *A* used in the computation.

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

*alpha* is the scalar  $\alpha$ .

Scope: **global**

Specified as: a number of the data type indicated in Table 61 on page 276.

*a* is the local part of the global triangular matrix *A*. This identifies the **first element** of the local array *A*. This subroutine computes the location of the first element of the local subarray used, based on *ia*, *ja*, *desc\_a*, *p*, *q*, *myrow*, and *mycol*; therefore, assuming the following:

If *side* = 'L', *numa* = *m*

If *side* = 'R', *numa* = *n*

the leading LOCp(*ia+numa-1*) by LOCq(*ja+numa-1*) part of the local array *A* must contain the local pieces of the leading *ia+numa-1* by *ja+numa-1* part of the global matrix, and:

- If *uplo* = 'U', the leading *numa*  $\times$  *numa* upper triangular part of the global triangular submatrix  $A_{ia:ia+numa-1, ja:ja+numa-1}$  must contain the upper triangular part of the submatrix, and the strictly lower triangular part is not referenced.

## PDTRMM and PZTRMM

- If  $uplo = 'L'$ , the leading  $numa \times numa$  lower triangular part of the global triangular submatrix  $A_{ia:ia+numa-1, ja:ja+numa-1}$  must contain the lower triangular part of the submatrix, and the strictly upper triangular part is not referenced.

**Note:** No data should be moved to form  $A^T$  or  $A^H$ ; that is, the matrix  $A$  should always be stored in its untransposed form.

Scope: **local**

Specified as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 61 on page 276. Details about the square block-cyclic data distribution of global matrix  $A$  are stored in  $desc_a$ .

$ia$  is the row index of the global matrix  $A$ , identifying the first row of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq M\_A$  and  $ia+numa-1 \leq M\_A$ .

$ja$  is the column index of the global matrix  $A$ , identifying the first column of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$  and  $ja+numa-1 \leq N\_A$ .

$desc_a$  is the array descriptor for global matrix  $A$ , described in the following table:

$desc\_a$	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $m = 0$ and $side = 'L'$ or $n = 0$ and $side = 'R'$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	If $m = 0$ and $side = 'L'$ or $n = 0$ and $side = 'R'$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_A < p$	Global
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < q$	Global
9	LLD_A	The leading dimension of the local array	$LLD\_A \geq \max(1, LOCp(M\_A))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.



$b$  is the local part of the global general matrix  $B$ . This identifies the **first element** of the local array  $B$ . This subroutine computes the location of the first element of the local subarray used, based on  $ib$ ,  $jb$ ,  $desc\_b$ ,  $p$ ,  $q$ ,  $myrow$ , and  $mycol$ ; therefore, the leading  $LOCp(ib+m-1)$  by  $LOCq(jb+n-1)$  part of the local array  $B$  must contain the local pieces of the leading  $ib+m-1$  by  $jb+n-1$  part of the global matrix.

Scope: **local**

Specified as: an  $LLD\_B$  by (at least)  $LOCq(N\_B)$  array, containing numbers of the data type indicated in Table 61 on page 276. Details about the block-cyclic data distribution of global matrix  $B$  are stored in  $desc\_b$ .

$ib$  is the row index of the global matrix  $B$ , identifying the first row of the submatrix  $B$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ib \leq M\_B$  and  $ib+m-1 \leq M\_B$ .

$jb$  is the column index of the global matrix  $B$ , identifying the first column of the submatrix  $B$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq jb \leq N\_B$  and  $jb+n-1 \leq N\_B$ .

$desc\_b$  is the array descriptor for global matrix  $B$ , described in the following table:

$desc\_b$	Name	Description	Limits	Scope
1	DTYPE_B	Descriptor type	DTYPE_B=1	Global
2	CTXT_B	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_B	Number of rows in the global matrix	If $m = 0$ or $n = 0$ : $M\_B \geq 0$ Otherwise: $M\_B \geq 1$	Global
4	N_B	Number of columns in the global matrix	If $m = 0$ or $n = 0$ : $N\_B \geq 0$ Otherwise: $N\_B \geq 1$	Global
5	MB_B	Row block size	$MB\_B \geq 1$	Global
6	NB_B	Column block size	$NB\_B \geq 1$	Global
7	RSRC_B	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_B < p$	Global
8	CSRC_B	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_B < q$	Global
9	LLD_B	The leading dimension of the local array	$LLD\_B \geq \max(1, LOCp(M\_B))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

**On Return**

$b$  is the updated local part of the global matrix  $B$ , containing the results of the computation.

Scope: **local**

Returned as: an LLD\_B by (at least) LOCq(N\_B) array, containing numbers of the data type indicated in Table 61 on page 276.

**Notes and Coding Rules**

1. These subroutines accept lowercase letters for the *side*, *uplo*, *transa*, and *diag* arguments.
2. For PDTRMM, if you specify 'C' for *transa*, it is interpreted as though you specified 'T'.
3. The matrices must have no common elements; otherwise, results are unpredictable.
4. PDTRMM and PZTRMM assume certain values in your array for parts of a triangular matrix. As a result, you do not have to set these values. For unit triangular matrices, the elements of the diagonal are assumed to be one. When using an upper or lower triangular matrix, the unreferenced elements in the lower and upper triangular part, respectively, are assumed to be zero.
5. The NUMROC utility subroutine can be used to determine the values of LOCp(M\_) and LOCq(N\_) used in the argument descriptions above. For details, see "Determining the Number of Rows and Columns in Your Local Arrays" on page 28 and "NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process" on page 868.
6. For suggested block sizes, see "Coding Tips for Optimizing Parallel Performance" on page 77.
7. The following values must be equal: CTXT\_A = CTXT\_B.
8. If  $A$  is **not** contained within a single block, that is, either of the following is true:
  - $numa + \text{mod}(ia-1, MB\_A) > MB\_A$
  - $numa + \text{mod}(ja-1, NB\_A) > NB\_A$
  - where:
    - If *side* = 'L',  $numa = m$
    - If *side* = 'R',  $numa = n$
 then:
  - The global triangular matrix  $A$  must be distributed using a square block-cyclic distribution; that is,  $MB\_A = NB\_A$ .
  - The global triangular matrix  $A$  must be aligned on a block boundary, that is:
    - $ia-1$  must be a multiple of  $MB\_A$ .
    - $ja-1$  must be a multiple of  $NB\_A$ .
9. If *side* = 'L':
  - If  $A$  is **not** contained within a single block, then:
    - The following block sizes must be equal:  $MB\_B = NB\_A$ .
    - The global matrix  $B$  must be aligned on a block row boundary; that is,  $ib-1$  must be a multiple of  $MB\_B$ .
  - In the process grid, the process row containing the first row of the submatrix  $A$  must also contain the first row of the submatrix  $B$ ; that is,  $iarow = ibrow$ , where:

- $irow = \text{mod}(((ia-1)/MB\_A)+RSRC\_A), p)$
  - $ibrow = \text{mod}(((ib-1)/MB\_B)+RSRC\_B), p)$
  - If  $A$  is contained within a single block, then  $B$  must be a block row matrix; that is, if  $p > 1$ :
    - $m+\text{mod}(ib-1, MB\_B) \leq MB\_B$
10. If  $side = 'R'$ :
- If  $A$  is **not** contained within a single block, then:
    - The following block sizes must be equal:  $NB\_B = MB\_A$
    - The global matrix  $B$  must be aligned on a block column boundary; that is,  $jb-1$  must be a multiple of  $NB\_B$ .
  - In the process grid, the process column containing the first column of the submatrix  $A$  must also contain the first column of the submatrix  $B$ , that is,  $iacol = ibcol$ , where:
    - $iacol = \text{mod}(((ja-1)/NB\_A)+CSRC\_A), q)$
    - $ibcol = \text{mod}(((jb-1)/NB\_B)+CSRC\_B), q)$
  - If  $A$  is contained within a single block, then  $B$  must be a block column matrix; that is, if  $q > 1$ :
    - $n+\text{mod}(jb-1, NB\_B) \leq NB\_B$

## Error Conditions

### Computational Errors

None

### Resource Errors

Unable to allocate work space

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1.  $DTYPE\_A$  is invalid.
2.  $DTYPE\_B$  is invalid.

#### Stage 2:

1.  $CTXT\_A$  is invalid.

#### Stage 3:

1. This subroutine was called from outside the process grid.

#### Stage 4:

1.  $side \neq 'L'$  or  $'R'$
2.  $uplo \neq 'U'$  or  $'L'$
3.  $transa \neq 'N', 'T',$  or  $'C'$
4.  $diag \neq 'N'$  or  $'U'$
5.  $m < 0$
6.  $n < 0$
7.  $M\_A < 0$  and  $m = 0$  and  $side = 'L'$ ;  $M\_A < 0$  and  $n = 0$  and  $side = 'R'$ ;  $M\_A < 1$  otherwise
8.  $N\_A < 0$  and  $m = 0$  and  $side = 'L'$ ;  $N\_A < 0$  and  $n = 0$  and  $side = 'R'$ ;  $N\_A < 1$  otherwise
9.  $MB\_A < 1$
10.  $NB\_A < 1$
11.  $M\_B < 0$  and  $(m = 0$  or  $n = 0)$ ;  $M\_B < 1$  otherwise
12.  $N\_B < 0$  and  $(m = 0$  or  $n = 0)$ ;  $N\_B < 1$  otherwise
13.  $MB\_B < 1$

14.  $NB\_B < 1$
15.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
16.  $CSRC\_A < 0$  or  $CSRC\_A \geq q$
17.  $RSRC\_B < 0$  or  $RSRC\_B \geq p$
18.  $CSRC\_B < 0$  or  $CSRC\_B \geq q$
19.  $ia < 1$
20.  $ja < 1$
21.  $ib < 1$
22.  $jb < 1$
23.  $CTXT\_A \neq CTXT\_B$

**Stage 5:**

1.  $MB\_A \neq NB\_A$

If  $A$  is **not** contained within a single block, that is, either of the following is true:

- $numa + \text{mod}(ia-1, MB\_A) > MB\_A$
- $numa + \text{mod}(ja-1, NB\_A) > NB\_A$
- where:
  - If  $side = 'L'$ ,  $numa = m$
  - If  $side = 'R'$ ,  $numa = n$

and:

2.  $side = 'L'$  and  $MB\_B \neq NB\_A$
3.  $side = 'R'$  and  $NB\_B \neq MB\_A$

If ( $m \neq 0$  or  $side \neq 'L'$ ) and ( $n \neq 0$  or  $side \neq 'R'$ ):

1.  $ia > M\_A$
2.  $ja > N\_A$
3.  $ia+numa-1 > M\_A$
4.  $ja+numa-1 > N\_A$

where  $numa = m$  if  $side = 'L'$  and  $numa = n$  if  $side = 'R'$ .

If  $m \neq 0$  and  $n \neq 0$ :

1.  $ib > M\_B$
2.  $jb > N\_B$
3.  $ib+m-1 > M\_B$
4.  $jb+n-1 > N\_B$

If  $A$  is not contained in a single block:

1.  $\text{mod}(ia-1, MB\_A) \neq 0$
2.  $\text{mod}(ja-1, NB\_A) \neq 0$
3.  $side = 'L'$  and  $\text{mod}(ib-1, MB\_B) \neq 0$
4.  $side = 'R'$  and  $\text{mod}(jb-1, NB\_B) \neq 0$

**Stage 6:**

1.  $LLD\_A < \max(1, \text{LOCp}(M\_A))$
2.  $LLD\_B < \max(1, \text{LOCp}(M\_B))$

If  $side = 'L'$ :

3. In the process grid, the process row containing the first row of the submatrix  $A$  does not contain the first row of the submatrix  $B$ ; that is,  $iarow \neq ibrow$ , where:
  - $iarow = \text{mod}((((ia-1)/MB\_A)+RSRC\_A), p)$
  - $ibrow = \text{mod}((((ib-1)/MB\_B)+RSRC\_B), p)$
4. If  $A$  is contained in a single block:
  - $p > 1$  and  $m + \text{mod}(ib-1, MB\_B) > MB\_B$

If  $side = 'R'$ :

5. In the process grid, the process column containing the first column of the submatrix  $A$  does not contain the first column of the submatrix  $B$ ; that is,  $iacol \neq ibcol$ , where:
  - $iacol = \text{mod}(((ja-1)/NB\_A)+CSRC\_A), q)$
  - $ibcol = \text{mod}(((jb-1)/NB\_B)+CSRC\_B), q)$
6. If  $A$  is contained in a single block:
  - $q > 1$  and  $n+\text{mod}(jb-1, NB\_B) > NB\_B$

## Examples

### Example 1

This example computes  $B = \alpha AB$  using a  $2 \times 2$  process grid.

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      SIDE  UPLO  TRANSA  DIAG  M  N  ALPHA  A  IA  JA  DESC_A
      |    |    |      |    |  |  |      |  |  |  |
CALL PDTRMM( 'L' , 'U' , 'N' , 'N' , 5 , 3 , 1.0D0 , A , 1 , 1 , DESC_A ,

      B  IB  JB  DESC_B
      |  |  |  |
      B , 1 , 1 , DESC_B )
```

	Desc_A	Desc_B
DTYPE_	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	5	5
N_	5	3
MB_	2	2
NB_	2	2
RSRC_	0	0
CSRC_	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>

#### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:
 

```
LLD_A = MAX(1, NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_B = MAX(1, NUMROC(M_B, MB_B, MYROW, RSRC_B, NPROW))
```

In this example,  $LLD\_A = LLD\_B = 3$  on  $P_{00}$  and  $P_{01}$ , and  $LLD\_A = LLD\_B = 2$  on  $P_{10}$  and  $P_{11}$ .

Global triangular matrix  $A$  of order 5 is upper triangular with block size  $2 \times 2$ :

```
B,D      0      1      2
0  [ 3.0 -1.0 | 2.0 2.0 | 1.0 ]
    [ .   -2.0 | 4.0 -1.0 | 3.0 ]
```

## PDTRMM and PZTRMM

1	-----	-----	-----
	. .	-3.0 0.0	2.0
	. .	. 4.0	-2.0
2	-----	-----	-----
	. .	. .	1.0

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	P <sub>00</sub>	P <sub>01</sub>
2	-----	-----
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for *A*:

p,q	0	1
0	3.0 -1.0 1.0	2.0 2.0
	. -2.0 3.0	4.0 -1.0
	. . 1.0	. .
1	. . 2.0	-3.0 0.0
	. . -2.0	. 4.0

Global rectangular  $5 \times 3$  matrix *B* with block size  $2 \times 2$ :

B,D	0	1
0	2.0 3.0	1.0
	5.0 5.0	4.0
1	0.0 1.0	2.0
	3.0 1.0	-3.0
2	-1.0 2.0	1.0

The following is the  $2 \times 2$  process grid:

B,D	0	1
0	P <sub>00</sub>	P <sub>01</sub>
2	-----	-----
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for *B*:

p,q	0	1
0	2.0 3.0	1.0
	5.0 5.0	4.0
	-1.0 2.0	1.0
1	0.0 1.0	2.0
	3.0 1.0	-3.0

**Output:**

Global rectangular  $5 \times 3$  matrix *B* with block size  $2 \times 2$ :

B,D	0	1
0	6.0 10.0	-2.0
	-16.0 -1.0	6.0

1	-----	-----
	-2.0 1.0	-4.0
	14.0 0.0	-14.0
2	-----	-----
	-1.0 2.0	1.0

The following is the  $2 \times 2$  process grid:

B,D	0	1
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for  $B$ :

p,q	0	1
0	6.0 10.0	-2.0
	-16.0 -1.0	6.0
	-1.0 2.0	1.0
1	-2.0 1.0	-4.0
	14.0 0.0	-14.0

## Example 2

This example computes  $B = \alpha AB$  using a  $2 \times 2$  process grid.

**Call Statements and Input:**

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      SIDE UPLO TRANSA DIAG M  N  ALPHA  A  IA  JA  DESC_A
CALL PZTRMM( 'L' , 'U' , 'C' , 'N' , 5 , 1 ,  ALPHA , A , 1 , 1 , DESC_A ,

      B  IB  JB  DESC_B
      |  |  |  |
      B , 1 , 1 , DESC_B )

ALPHA = (1.0, 0.0)
```

	Desc_A	Desc_B
DTYPE_	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	5	5
N_	5	1
MB_	2	2
NB_	2	2
RSRC_	0	0
CSRC_	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:  
 $LLD\_A = \text{MAX}(1, \text{NUMROC}(M\_A, MB\_A, MYROW, RSRC\_A, NPROW))$   
 $LLD\_B = \text{MAX}(1, \text{NUMROC}(M\_B, MB\_B, MYROW, RSRC\_B, NPROW))$   
 In this example,  $LLD\_A = LLD\_B = 3$  on  $P_{00}$  and  $P_{01}$ , and  
 $LLD\_A = LLD\_B = 2$  on  $P_{10}$  and  $P_{11}$ .

Global triangular matrix  $A$  of order 5 is upper triangular with block size  $2 \times 2$ :

B,D	0	1	2
0	$\begin{pmatrix} (-4.0, 1.0) & (4.0, -3.0) \\ . & (-2.0, 0.0) \end{pmatrix}$	$\begin{pmatrix} (-1.0, 3.0) & (0.0, 0.0) \\ (-3.0, -1.0) & (-2.0, -1.0) \end{pmatrix}$	$\begin{pmatrix} (-1.0, 0.0) \\ (4.0, 3.0) \end{pmatrix}$
1	$\begin{pmatrix} . & . \\ . & . \end{pmatrix}$	$\begin{pmatrix} (-5.0, 3.0) & (-3.0, -3.0) \\ . & (4.0, -4.0) \end{pmatrix}$	$\begin{pmatrix} (-5.0, -5.0) \\ (2.0, 0.0) \end{pmatrix}$
2	$\begin{pmatrix} . & . \end{pmatrix}$	$\begin{pmatrix} . & . \end{pmatrix}$	$\begin{pmatrix} (2.0, -1.0) \end{pmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

Local arrays for  $A$ :

p,q	0	1
0	$\begin{pmatrix} (-4.0, 1.0) & (4.0, -3.0) \\ . & (-2.0, 0.0) \\ . & . \end{pmatrix}$	$\begin{pmatrix} (-1.0, 3.0) & (0.0, 0.0) \\ (-3.0, -1.0) & (-2.0, -1.0) \\ . & . \end{pmatrix}$
1	$\begin{pmatrix} . & . \\ . & . \end{pmatrix}$	$\begin{pmatrix} (-5.0, -5.0) & (-5.0, 3.0) \\ (2.0, 0.0) & (-3.0, -3.0) \\ . & (4.0, -4.0) \end{pmatrix}$

Global rectangular  $5 \times 1$  matrix  $B$  with block size  $2 \times 2$ :

B,D	0
0	$\begin{pmatrix} (3.0, 4.0) \\ (-4.0, 2.0) \end{pmatrix}$
1	$\begin{pmatrix} (-5.0, 0.0) \\ (1.0, 3.0) \end{pmatrix}$
2	$\begin{pmatrix} (3.0, 1.0) \end{pmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0	--
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

Local arrays for  $B$ :



p,q	0	1
0	( 3.0, 4.0)	.
	(-4.0, 2.0)	.
	( 3.0, 1.0)	.
1	(-5.0, 0.0)	.
	( 1.0, 3.0)	.

**Output:**

Global rectangular  $5 \times 1$  matrix  $B$  with block size  $2 \times 2$ :

B,D	0
0	(-8.0,-19.0)
	( 8.0, 21.0)
1	(44.0, -8.0)
	(13.0, -7.0)
2	(19.0, 2.0)

The following is the  $2 \times 2$  process grid:

B,D	0	--
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for  $B$ :

p,q	0	1
0	(-8.0,-19.0)	.
	( 8.0, 21.0)	.
	(19.0, 2.0)	.
1	(44.0, -8.0)	.
	(13.0, -7.0)	.

## PDTRSM and PZTRSM — Solution of Triangular System of Equations with Multiple Right-Hand Sides

### Purpose

PDTRSM perform one of the following solves for a triangular system of equations with multiple right-hand sides, using scalar  $\alpha$ , rectangular matrix  $B$ , and triangular matrix  $A$  or its transpose:

Solution	Equation
1. $B \leftarrow \alpha(A^{-1})B$	$AX = \alpha B$
2. $B \leftarrow \alpha(A^{-T})B$	$A^T X = \alpha B$
3. $B \leftarrow \alpha B(A^{-1})$	$XA = \alpha B$
4. $B \leftarrow \alpha B(A^{-T})$	$XA^T = \alpha B$

PZTRSM performs one of the following solves for a triangular system of equations with multiple right-hand sides, using scalar  $\alpha$ , rectangular matrix  $B$ , and triangular matrix  $A$ , its transpose, or its conjugate transpose:

Solution	Equation
1. $B \leftarrow \alpha(A^{-1})B$	$AX = \alpha B$
2. $B \leftarrow \alpha(A^{-T})B$	$A^T X = \alpha B$
3. $B \leftarrow \alpha B(A^{-1})$	$XA = \alpha B$
4. $B \leftarrow \alpha B(A^{-T})$	$XA^T = \alpha B$
5. $B \leftarrow \alpha(A^{-H})B$	$A^H X = \alpha B$
6. $B \leftarrow \alpha B(A^{-H})$	$XA^H = \alpha B$

where, in the formulas above:

- $A$  represents the global triangular submatrix:
  - For  $side = 'L'$ , it is  $A_{ia:ia+m-1, ja:ja+m-1}$ .
  - For  $side = 'R'$ , it is  $A_{ia:ia+n-1, ja:ja+n-1}$ .
- $B$  represents the global general submatrix  $B_{ib:ib+m-1, jb:jb+n-1}$ .
- $\alpha$  is a scalar.

### Notes:

1. The term  $X$  used in the systems of equations listed above represents the output solution matrix. It is important to note that, in this subroutine, the solution matrix is actually returned in the input-output argument  $b$ .
2. No data should be moved to form  $A^T$  or  $A^H$ ; that is, the matrix  $A$  should always be stored in its untransposed form.

If  $m = 0$  or  $n = 0$ , no computation is performed, and the subroutine returns after doing some parameter checking.

See references [15] and [16].

Table 62. Data Types

$\alpha, A, B$	Subprogram
Long-precision real	PDTRSM
Long-precision complex	PZTRSM

## Syntax

<b>Fortran</b>	CALL PDTRSM   PZTRSM ( <i>side</i> , <i>uplo</i> , <i>transa</i> , <i>diag</i> , <i>m</i> , <i>n</i> , <i>alpha</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>b</i> , <i>ib</i> , <i>jb</i> , <i>desc_b</i> )
<b>C and C++</b>	pdtrsm   pztrsm ( <i>side</i> , <i>uplo</i> , <i>transa</i> , <i>diag</i> , <i>m</i> , <i>n</i> , <i>alpha</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>b</i> , <i>ib</i> , <i>jb</i> , <i>desc_b</i> );

### On Entry

*side* indicates whether *A* is located to the left or right of *B* in the system of equations, where:

If *side* = 'L', *A* is to the left of *B*.

If *side* = 'R', *A* is to the right of *B*.

Scope: **global**

Specified as: a single character; *side* = 'L' or 'R'.

*uplo* indicates whether the upper or lower triangular part of the global triangular submatrix *A* is referenced, where:

If *uplo* = 'U', the upper triangular part is referenced.

If *uplo* = 'L', the lower triangular part is referenced.

Scope: **global**

Specified as: a single character; *uplo* = 'U' or 'L'.

*transa* indicates the form of matrix *A* used in the system of equations, where:

If *transa* = 'N', *A* is used.

If *transa* = 'T',  $A^T$  is used.

If *transa* = 'C',  $A^H$  is used.

Scope: **global**

Specified as: a single character; *transa* = 'N', 'T', or 'C'.

*diag* indicates the characteristics of the diagonal of matrix *A*, where:

If *diag* = 'U', *A* is a unit triangular matrix.

If *diag* = 'N', *A* is not a unit triangular matrix.

Scope: **global**

Specified as: a single character; *diag* = 'U' or 'N'.

*m* is the number of rows in submatrix *B*, and:

If *side* = 'L', it is the number of rows and columns in submatrix *A* used in the computation.

Scope: **global**

Specified as: a fullword integer;  $m \geq 0$ .

*n* is the number of columns in submatrix *B*, and:

If *side* = 'R', it is the number of rows and columns in submatrix *A* used in the computation.

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

*alpha* is the scalar  $\alpha$ .

Scope: **global**

Specified as: a number of the data type indicated in Table 62 on page 288.

*a* is the local part of the global triangular matrix *A*, used in the system of equations. This identifies the **first element** of the local array *A*. This subroutine computes the location of the first element of the local subarray used, based on *ia*, *ja*, *desc\_a*, *p*, *q*, *myrow*, and *mycol*; therefore, assuming the following:

If *side* = 'L', *numa* = *m*

If *side* = 'R', *numa* = *n*

the leading LOCp(*ia+numa*-1) by LOCq(*ja+numa*-1) part of the local array *A* must contain the local pieces of the leading *ia+numa*-1 by *ja+numa*-1 part of the global matrix, and:

- If *uplo* = 'U', the leading *numa* × *numa* upper triangular part of the global triangular submatrix  $A_{ia:ia+numa-1, ja:ja+numa-1}$  must contain the upper triangular part of the submatrix, and the strictly lower triangular part is not referenced.
- If *uplo* = 'L', the leading *numa* × *numa* lower triangular part of the global triangular submatrix  $A_{ia:ia+numa-1, ja:ja+numa-1}$  must contain the lower triangular part of the submatrix, and the strictly upper triangular part is not referenced.

**Note:** No data should be moved to form  $A^T$  or  $A^H$ ; that is, the matrix *A* should always be stored in its untransposed form.

Scope: **local**

Specified as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 62 on page 288. Details about the block-cyclic data distribution of global matrix *A* are stored in *desc\_a*.

*ia* is the row index of the global matrix *A*, identifying the first row of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq M\_A$  and  $ia+numa-1 \leq M\_A$ .

*ja* is the column index of the global matrix *A*, identifying the first column of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$  and  $ja+numa-1 \leq N\_A$ .

*desc\_a* is the array descriptor for global matrix *A*, described in the following table:

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global

<i>desc_a</i>	Name	Description	Limits	Scope
3	M_A	Number of rows in the global matrix	If <i>side</i> = 'L' and $m = 0$ : M_A $\geq 0$ If <i>side</i> = 'R' and $n = 0$ : M_A $\geq 0$ Otherwise: M_A $\geq 1$	Global
4	N_A	Number of columns in the global matrix	If <i>side</i> = 'L' and $m = 0$ : N_A $\geq 0$ If <i>side</i> = 'R' and $n = 0$ : N_A $\geq 0$ Otherwise: N_A $\geq 1$	Global
5	MB_A	Row block size	MB_A $\geq 1$	Global
6	NB_A	Column block size	NB_A $\geq 1$	Global
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq \text{RSRC\_A} < p$	Global
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq \text{CSRC\_A} < q$	Global
9	LLD_A	The leading dimension of the local array	LLD_A $\geq \max(1, \text{LOCp}(\text{M\_A}))$	Local

Specified as: an array of (at least) length 9, containing fullword integers.

*b* is the local part of the global general matrix *B*, containing the right-hand sides of the triangular system to be solved. This identifies the **first element** of the local array B. This subroutine computes the location of the first element of the local subarray used, based on *ib*, *jb*, *desc\_b*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading LOCp(*ib+m-1*) by LOCq(*jb+n-1*) part of the local array B must contain the local pieces of the leading *ib+m-1* by *jb+n-1* part of the global matrix.

Scope: **local**

Specified as: an LLD\_B by (at least) LOCq(N\_B) array, containing numbers of the data type indicated in Table 62 on page 288. Details about the block-cyclic data distribution of global matrix *B* are stored in *desc\_b*.

*ib* is the row index of the global matrix *B*, identifying the first row of the submatrix *B*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ib \leq \text{M\_B}$  and  $ib+m-1 \leq \text{M\_B}$ .

*jb* is the column index of the global matrix *B*, identifying the first column of the submatrix *B*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq jb \leq \text{N\_B}$  and  $jb+n-1 \leq \text{N\_B}$ .

*desc\_b* is the array descriptor for global matrix *B*, described in the following table:

<i>desc_b</i>	Name	Description	Limits	Scope
1	DTYPE_B	Descriptor type	DTYPE_B=1	Global

## PDTRSM and PZTRSM

<i>desc_b</i>	Name	Description	Limits	Scope
2	CTXT_B	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_B	Number of rows in the global matrix	If <i>side</i> = 'L' and <i>m</i> = 0: M_B ≥ 0 If <i>side</i> = 'R' and <i>n</i> = 0: M_B ≥ 0 Otherwise: M_B ≥ 1	Global
4	N_B	Number of columns in the global matrix	N_B ≥ 1	Global
5	MB_B	Row block size	MB_B ≥ 1	Global
6	NB_B	Column block size	If <i>side</i> = 'L' and <i>m</i> = 0: N_B ≥ 0 If <i>side</i> = 'R' and <i>n</i> = 0: N_B ≥ 0 Otherwise: N_B ≥ 1	Global
7	RSRC_B	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq \text{RSRC\_B} < p$	Global
8	CSRC_B	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq \text{CSRC\_B} < q$	Global
9	LLD_B	The leading dimension of the local array	LLD_B ≥ max(1,LOCp(M_B))	Local

Specified as: an array of (at least) length 9, containing fullword integers.

### On Return

*b* is the updated local part of the global matrix *B*, containing the *n* solution vectors of length *m*.

Scope: **local**

Returned as: an LLD\_B by (at least) LOCq(N\_B) array, containing numbers of the data type indicated in Table 62 on page 288.

## Notes and Coding Rules

1. These subroutines accept lowercase letters for the *side*, *uplo*, *transa*, and *diag* arguments.
2. For PDTRSM, if you specify 'C' for *transa*, it is interpreted as though you specified 'T'.
3. The matrices must have no common elements; otherwise, results are unpredictable.
4. PDTRSM and PZTRSM assume certain values in your array for parts of a triangular matrix. As a result, you do not have to set these values. For unit triangular matrices, the elements of the diagonal are assumed to be one. When using an upper or lower triangular matrix, the unreferenced elements in the lower and upper triangular part, respectively, are assumed to be zero.

5. The NUMROC utility subroutine can be used to determine the values of  $LOCp(M\_)$  and  $LOCq(N\_)$  used in the argument descriptions above. For details, see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28 and “NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process” on page 868.
6. For suggested block sizes, see “Coding Tips for Optimizing Parallel Performance” on page 77.
7. The following values must be equal:  $CTXT\_A = CTXT\_B$ .
8. If  $A$  is **not** contained within a single block, that is, either of the following is true:
  - $numa + \text{mod}(ia-1, MB\_A) > MB\_A$
  - $numa + \text{mod}(ja-1, NB\_A) > NB\_A$
  - where:
    - If  $side = 'L'$ ,  $numa = m$
    - If  $side = 'R'$ ,  $numa = n$

then the global triangular matrix  $A$  must be distributed using a square block-cyclic distribution; that is,  $MB\_A = NB\_A$ .

9. If  $A$  is **not** contained within a single block, then the global triangular matrix  $A$  must be aligned on a block boundary, that is:
  - $ia-1$  must be a multiple of  $MB\_A$ .
  - $ja-1$  must be a multiple of  $NB\_A$ .
10. If  $side = 'L'$ :
  - If  $A$  is **not** contained within a single block, then:
    - The following block sizes must be equal:  $MB\_B = NB\_A$
    - The global matrix  $B$  must be aligned on a block row boundary; that is,  $ib-1$  must be a multiple of  $MB\_B$ .
  - In the process grid, the process row containing the first row of the submatrix  $A$  must also contain the first row of the submatrix  $B$ ; that is,  $iarow = ibrow$ , where:
    - $iarow = \text{mod}((((ia-1)/MB\_A)+RSRC\_A), p)$
    - $ibrow = \text{mod}((((ib-1)/MB\_B)+RSRC\_B), p)$
11. If  $side = 'R'$ :
  - If  $A$  is **not** contained within a single block, then:
    - The following block sizes must be equal:  $NB\_B = MB\_A$
    - The global matrix  $B$  must be aligned on a block column boundary; that is,  $jb-1$  must be a multiple of  $NB\_B$ .
  - In the process grid, the process column containing the first column of the submatrix  $A$  must also contain the first column of the submatrix  $B$ , that is,  $iacol = ibcol$ , where:
    - $iacol = \text{mod}((((ja-1)/NB\_A)+CSRC\_A), q)$
    - $ibcol = \text{mod}((((jb-1)/NB\_B)+CSRC\_B), q)$
12. If  $A$  is contained within a single block, then:
  - If  $side = 'L'$ , then  $B$  must be a block row matrix; that is, if  $p > 1$ :
    - $m + \text{mod}(ib-1, MB\_B) \leq MB\_B$
  - If  $side = 'R'$ , then  $B$  must be a block column matrix; that is, if  $q > 1$ :
    - $n + \text{mod}(jb-1, NB\_B) \leq NB\_B$

## Error Conditions

### Computational Errors

None

### Resource Errors

Unable to allocate work space

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. DTYPE\_A is invalid.
2. DTYPE\_B is invalid.

#### Stage 2:

1. CTXT\_A is invalid.

#### Stage 3:

1. This subroutine was called from outside the process grid.

#### Stage 4:

1. *side*  $\neq$  'L' or 'R'
2. *uplo*  $\neq$  'U' or 'L'
3. *transa*  $\neq$  'N', 'T', or 'C'
4. *diag*  $\neq$  'N' or 'U'
5. *m* < 0
6. *n* < 0
7. *M\_A* < 0 and *m* = 0 and *side* = 'L'; *M\_A* < 0 and *n* = 0 and *side* = 'R';  
  *M\_A* < 1 otherwise
8. *N\_A* < 0 and *m* = 0 and *side* = 'L'; *N\_A* < 0 and *n* = 0 and *side* = 'R';  
  *N\_A* < 1 otherwise
9. *MB\_A* < 1
10. *NB\_A* < 1
11. *M\_B* < 0 and (*m* = 0 or *n* = 0); *M\_B* < 1 otherwise
12. *N\_B* < 0 and (*m* = 0 or *n* = 0); *N\_B* < 1 otherwise
13. *MB\_B* < 1
14. *NB\_B* < 1
15. *RSRC\_A* < 0 or *RSRC\_A*  $\geq p$
16. *CSRC\_A* < 0 or *CSRC\_A*  $\geq q$
17. *RSRC\_B* < 0 or *RSRC\_B*  $\geq p$
18. *CSRC\_B* < 0 or *CSRC\_B*  $\geq q$
19. *ia* < 1
20. *ja* < 1
21. *ib* < 1
22. *jb* < 1
23. CTXT\_A  $\neq$  CTXT\_B

**Stage 5:** If *A* is **not** contained within a single block, that is, either of the following is true:

- *numa*+mod(*ia*-1, *MB\_A*) > *MB\_A*
- *numa*+mod(*ja*-1, *NB\_A*) > *NB\_A*
- where:
  - If *side* = 'L', *numa* = *m*
  - If *side* = 'R', *numa* = *n*

then:



1.  $MB\_A \neq NB\_A$
2.  $side = 'L'$  and  $MB\_B \neq NB\_A$
3.  $side = 'R'$  and  $NB\_B \neq MB\_A$

If ( $m \neq 0$  or  $side \neq 'L'$ ) and ( $n \neq 0$  or  $side \neq 'R'$ ):

1.  $ia > M\_A$
2.  $ja > N\_A$
3.  $ia+numa-1 > M\_A$
4.  $ja+numa-1 > N\_A$

where  $numa = m$  if  $side = 'L'$  and  $numa = n$  if  $side = 'R'$ .

If  $m \neq 0$  and  $n \neq 0$ :

1.  $ib > M\_B$
2.  $jb > N\_B$
3.  $ib+m-1 > M\_B$
4.  $jb+n-1 > N\_B$

If  $A$  is not contained in a single block:

1.  $\text{mod}(ia-1, MB\_A) \neq 0$
2.  $\text{mod}(ja-1, NB\_A) \neq 0$
3.  $side = 'L'$  and  $\text{mod}(ib-1, MB\_B) \neq 0$
4.  $side = 'R'$  and  $\text{mod}(jb-1, NB\_B) \neq 0$

**Stage 6:**

1.  $LLD\_A < \max(1, LOCp(M\_A))$
2.  $LLD\_B < \max(1, LOCp(M\_B))$

If  $side = 'L'$ :

1. In the process grid, the process row containing the first row of the submatrix  $A$  does not contain the first row of the submatrix  $B$ ; that is,  $iarow \neq ibrow$ , where:
  - $iarow = \text{mod}((((ia-1)/MB\_A)+RSRC\_A), p)$
  - $ibrow = \text{mod}((((ib-1)/MB\_B)+RSRC\_B), p)$
2. If  $A$  is contained in a single block:
  - $p > 1$  and  $m+\text{mod}(ib-1, MB\_B) > MB\_B$

If  $side = 'R'$ :

1. In the process grid, the process column containing the first column of the submatrix  $A$  does not contain the first column of the submatrix  $B$ ; that is,  $iacol \neq ibcol$ , where:
  - $iacol = \text{mod}((((ja-1)/NB\_A)+CSRC\_A), q)$
  - $ibcol = \text{mod}((((jb-1)/NB\_B)+CSRC\_B), q)$
2. If  $A$  is contained in a single block:
  - $q > 1$  and  $n+\text{mod}(jb-1, NB\_B) > NB\_B$

## Examples

### Example 1

This example shows the solution  $B \leftarrow \alpha(A^{-1})B$  using a  $2 \times 2$  process grid.

## PDTRSM and PZTRSM

### Call Statements and Input:

```

ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      SIDE  UPLO  TRANSA  DIAG  M  N  ALPHA  A  IA  JA  DESC_A
      |      |      |      |      |  |      |  |  |  |
CALL PDTRSM( 'L' , 'U' , 'N' , 'N' , 5 , 3 , 1.0D0 , A , 1 , 1 , DESC_A ,

      B  IB  JB  DESC_B
      |  |  |  |
      B , 1 , 1 , DESC_B )

```

	Desc_A	Desc_B
DTYPE_	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	5	5
N_	5	3
MB_	2	2
NB_	2	2
RSRC_	0	0
CSRC_	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>

#### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:  
 $LLD\_A = \text{MAX}(1, \text{NUMROC}(M\_A, MB\_A, MYROW, RSRC\_A, NPROW))$   
 $LLD\_B = \text{MAX}(1, \text{NUMROC}(M\_B, MB\_B, MYROW, RSRC\_B, NPROW))$   
 In this example,  $LLD\_A = LLD\_B = 3$  on  $P_{00}$  and  $P_{01}$ , and  
 $LLD\_A = LLD\_B = 2$  on  $P_{10}$  and  $P_{11}$ .

Global triangular matrix *A* of order 5 is upper triangular with block size  $2 \times 2$ :

$$\begin{array}{c|ccc}
 \text{B,D} & 0 & 1 & 2 \\
 \hline
 0 & \begin{bmatrix} 3.0 & -1.0 \\ . & -2.0 \end{bmatrix} & \begin{bmatrix} 2.0 & 2.0 \\ 4.0 & -1.0 \end{bmatrix} & \begin{bmatrix} 1.0 \\ 3.0 \end{bmatrix} \\
 1 & \begin{bmatrix} . & . \\ . & . \end{bmatrix} & \begin{bmatrix} -3.0 & 0.0 \\ . & 4.0 \end{bmatrix} & \begin{bmatrix} 2.0 \\ -2.0 \end{bmatrix} \\
 2 & \begin{bmatrix} . & . \end{bmatrix} & \begin{bmatrix} . & . \end{bmatrix} & \begin{bmatrix} 1.0 \end{bmatrix}
 \end{array}$$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

Local arrays for  $A$ :

p,q	0	1
0	$\begin{bmatrix} 3.0 & -1.0 & 1.0 \\ . & -2.0 & 3.0 \\ . & . & 1.0 \end{bmatrix}$	$\begin{bmatrix} 2.0 & 2.0 \\ 4.0 & -1.0 \\ . & . \end{bmatrix}$
1	$\begin{bmatrix} . & . & 2.0 \\ . & . & -2.0 \end{bmatrix}$	$\begin{bmatrix} -3.0 & 0.0 \\ . & 4.0 \end{bmatrix}$

Global general  $5 \times 3$  matrix  $B$  with block size  $2 \times 2$ :

B,D	0	1
0	$\begin{bmatrix} 6.0 & 10.0 \\ -16.0 & -1.0 \end{bmatrix}$	$\begin{bmatrix} -2.0 \\ 6.0 \end{bmatrix}$
1	$\begin{bmatrix} -2.0 & 1.0 \\ 14.0 & 0.0 \end{bmatrix}$	$\begin{bmatrix} -4.0 \\ -14.0 \end{bmatrix}$
2	$\begin{bmatrix} -1.0 & 2.0 \end{bmatrix}$	$\begin{bmatrix} 1.0 \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0	1
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$

Local arrays for  $B$ :

p,q	0	1
0	$\begin{bmatrix} 6.0 & 10.0 \\ -16.0 & -1.0 \\ -1.0 & 2.0 \end{bmatrix}$	$\begin{bmatrix} -2.0 \\ 6.0 \\ 1.0 \end{bmatrix}$
1	$\begin{bmatrix} -2.0 & 1.0 \\ 14.0 & 0.0 \end{bmatrix}$	$\begin{bmatrix} -4.0 \\ -14.0 \end{bmatrix}$

**Output:**

Global general  $5 \times 3$  matrix  $B$  with block size  $2 \times 2$ :

B,D	0	1
0	$\begin{bmatrix} 2.0 & 3.0 \\ 5.0 & 5.0 \end{bmatrix}$	$\begin{bmatrix} 1.0 \\ 4.0 \end{bmatrix}$
1	$\begin{bmatrix} 0.0 & 1.0 \\ 3.0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 2.0 \\ -3.0 \end{bmatrix}$
2	$\begin{bmatrix} -1.0 & 2.0 \end{bmatrix}$	$\begin{bmatrix} 1.0 \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0	1
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$

## PDTRSM and PZTRSM

Local arrays for  $B$ :

p,q	0	1
0	2.0 3.0 5.0 5.0 -1.0 2.0	1.0 4.0 1.0
1	0.0 1.0 3.0 1.0	2.0 -3.0

### Example 2

This example shows the solution  $B \leftarrow \alpha(A^{-H})B$  using a  $2 \times 2$  process grid.

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      SIDE UPLO  TRANSA  DIAG  M  N  ALPHA  A  IA  JA  DESC_A
      |    |    |      |  |  |  |    |  |  |  |
CALL PZTRSM( 'L' , 'U' , 'C' , 'N' , 5 , 2 , ALPHA , A , 1 , 1 , DESC_A ,

      B  IB  JB  DESC_B
      |  |  |  |
      B , 1 , 1 , DESC_B )

ALPHA = (1.0D0, -1.0D0)
```

	Desc_A	Desc_B
DTYPE_	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	5	5
N_	5	2
MB_	2	2
NB_	2	2
RSRC_	0	0
CSRC_	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>

#### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.

2. Each process should set the LLD\_ as follows:

```
LLD_A = MAX(1, NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_B = MAX(1, NUMROC(M_B, MB_B, MYROW, RSRC_B, NPROW))
```

In this example, LLD\_A = LLD\_B = 3 on P<sub>00</sub> and P<sub>01</sub>, and  
LLD\_A = LLD\_B = 2 on P<sub>10</sub> and P<sub>11</sub>.

Global triangular matrix  $A$  of order 5 is upper triangular with block size  $2 \times 2$ :

B,D                      0                      1                      2

$$0 \quad \left[ \begin{array}{cc|cc|cc} (-4.0, 1.0) & (4.0, -3.0) & (-1.0, 3.0) & (0.0, 0.0) & (-1.0, 0.0) & \end{array} \right]$$

$$\begin{array}{c}
 1 \\
 2
 \end{array}
 \left[ \begin{array}{c|c|c}
 \begin{array}{cc}
 \cdot & (-2.0, 0.0) \\
 \hline
 \cdot & \cdot \\
 \cdot & \cdot \\
 \hline
 \cdot & \cdot
 \end{array}
 &
 \begin{array}{cc}
 (-3.0, -1.0) & (-2.0, -1.0) \\
 \hline
 (-5.0, 3.0) & (-3.0, -3.0) \\
 \cdot & (4.0, -4.0) \\
 \hline
 \cdot & \cdot
 \end{array}
 &
 \begin{array}{c}
 (4.0, 3.0) \\
 \hline
 (-5.0, -5.0) \\
 (2.0, 0.0) \\
 \hline
 (2.0, -1.0)
 \end{array}
 \end{array} \right]$$

The following is the  $2 \times 2$  process grid:

$$\begin{array}{c|c|c}
 \text{B,D} & 0 & 1 \\
 \hline
 0 & P_{00} & P_{01} \\
 2 & & \\
 \hline
 1 & P_{10} & P_{11}
 \end{array}$$

Local arrays for  $A$ :

$$\begin{array}{c|c|c}
 \text{p,q} & 0 & 1 \\
 \hline
 0 & \begin{array}{cc}
 (-4.0, 1.0) & (4.0, -3.0) \\
 \cdot & (-2.0, 0.0) \\
 \cdot & \cdot
 \end{array} & \begin{array}{cc}
 (-1.0, 0.0) & (4.0, 3.0) \\
 (-1.0, 3.0) & (0.0, 0.0) \\
 (-3.0, -1.0) & (-2.0, -1.0)
 \end{array} \\
 1 & \begin{array}{cc}
 \cdot & \cdot \\
 \cdot & \cdot
 \end{array} & \begin{array}{cc}
 (-5.0, -5.0) & (-5.0, 3.0) \\
 (2.0, 0.0) & (-3.0, -3.0) \\
 \cdot & (4.0, -4.0)
 \end{array}
 \end{array}$$

Global general  $5 \times 2$  matrix  $B$  with block size  $2 \times 2$ :

$$\begin{array}{c}
 \text{B,D} \\
 0 \\
 0 \\
 1 \\
 2
 \end{array}
 \left[ \begin{array}{cc}
 (5.5, -13.5) & (-3.0, -5.0) \\
 (-6.5, 14.5) & (-3.0, 5.0) \\
 \hline
 (26.0, 18.0) & (4.0, -3.0) \\
 (10.0, 3.0) & (6.0, -6.0) \\
 \hline
 (8.5, 10.5) & (13.0, -12.0)
 \end{array} \right]$$

The following is the  $2 \times 2$  process grid:

$$\begin{array}{c|c|c}
 \text{B,D} & 0 & -- \\
 \hline
 0 & P_{00} & P_{01} \\
 2 & & \\
 \hline
 1 & P_{10} & P_{11}
 \end{array}$$

Local arrays for  $B$ :

$$\begin{array}{c|c|c}
 \text{p,q} & 0 & 1 \\
 \hline
 0 & \begin{array}{cc}
 (5.5, -13.5) & (-3.0, -5.0) \\
 (-6.5, 14.5) & (-3.0, 5.0) \\
 (8.5, 10.5) & (13.0, -12.0)
 \end{array} & \begin{array}{c}
 \cdot \\
 \cdot \\
 \cdot
 \end{array} \\
 1 & \begin{array}{cc}
 (26.0, 18.0) & (4.0, -3.0) \\
 (10.0, 3.0) & (6.0, -6.0)
 \end{array} & \begin{array}{c}
 \cdot \\
 \cdot
 \end{array}
 \end{array}$$

**Output:**

Global general  $5 \times 2$  matrix  $B$  with block size  $2 \times 2$ :

$$\begin{array}{c}
 \text{B,D} \\
 0 \\
 0
 \end{array}
 \left[ \begin{array}{cc}
 (3.0, 4.0) & (2.0, 0.0)
 \end{array} \right]$$

## PDTRSM and PZTRSM

$$\begin{array}{c|c} & \begin{array}{c} (-4.0, 2.0) \quad (3.0, -1.0) \\ \hline (-5.0, 0.0) \quad (-1.0, 2.0) \\ (1.0, 3.0) \quad (0.0, -2.0) \\ \hline (3.0, 1.0) \quad (1.0, 3.0) \end{array} \\ \begin{array}{c} 1 \\ 2 \end{array} & \end{array}$$

The following is the  $2 \times 2$  process grid:

B,D	0	--
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

Local arrays for  $B$ :

p,q	0	1
0	$\begin{pmatrix} 3.0 & 4.0 \\ -4.0 & 2.0 \\ 3.0 & 1.0 \end{pmatrix} \begin{pmatrix} 2.0 & 0.0 \\ 3.0 & -1.0 \\ 1.0 & 3.0 \end{pmatrix}$	$\begin{pmatrix} . \\ . \\ . \end{pmatrix}$
1	$\begin{pmatrix} -5.0 & 0.0 \\ 1.0 & 3.0 \end{pmatrix} \begin{pmatrix} -1.0 & 2.0 \\ 0.0 & -2.0 \end{pmatrix}$	$\begin{pmatrix} . \\ . \end{pmatrix}$

## PDSYRK, PZSYRK, and PZHERK — Rank-K Update of a Real or Complex Symmetric or a Complex Hermitian Matrix

### Purpose

PDSYRK and PZSYRK compute one of the following rank-k updates:

- 1.  $C \leftarrow \alpha A A^T + \beta C$
- 2.  $C \leftarrow \alpha A^T A + \beta C$

PZHERK computes one of the following rank-k updates:

- 3.  $C \leftarrow \alpha A A^H + \beta C$
- 4.  $C \leftarrow \alpha A^H A + \beta C$

where, in the formulas above:

- $A$  represents the global general submatrix:
  - For  $trans = 'N'$ , it is  $A_{ia:ia+n-1, ja:ja+k-1}$ .
  - For  $trans = 'T'$  or  $'C'$ , it is  $A_{ia:ia+k-1, ja:ja+n-1}$ .
- $C$  represents the global submatrix  $C_{ic:ic+n-1, jc:jc+n-1}$ .

and:

- For PDSYRK, submatrix  $C$  is real symmetric.
- For PZSYRK, submatrix  $C$  is complex symmetric.
- For PZHERK, submatrix  $C$  is complex Hermitian.

**Note:** No data should be moved to form  $A^T$  or  $A^H$ ; that is, the matrix  $A$  should always be stored in its untransposed form.

In the following two cases, no computation is performed and the subroutine returns after doing some parameter checking:

- $n = 0$
- $\beta$  is one, and  $\alpha$  is zero or  $k = 0$ .

See references [15] and [16].

Table 63. Data Types

$A, C$	$\alpha, \beta$	Subprogram
Long-precision real	Long-precision real	PDSYRK
Long-precision complex	Long-precision complex	PZSYRK
Long-precision complex	Long-precision real	PZHERK

### Syntax

<b>Fortran</b>	CALL PDSYRK   PZSYRK   PZHERK ( <i>uplo, trans, n, k, alpha, a, ia, ja, desc_a, beta, c, ic, jc, desc_c</i> )
<b>C and C++</b>	pdsyrk   pzsyrk   pzherk ( <i>uplo, trans, n, k, alpha, a, ia, ja, desc_a, beta, c, ic, jc, desc_c</i> );

### On Entry

*uplo* indicates whether the upper or lower triangular part of the global submatrix  $C$  is referenced, where:

If *uplo* = 'U', the upper triangular part is referenced.

If *uplo* = 'L', the lower triangular part is referenced.

Scope: **global**

## PDSYRK, PZSYRK, and PZHERK

- Specified as: a single character; *uplo* = 'U' or 'L'.
- trans* indicates which computation is performed, where:
- If *trans* = 'N',  $A$  is used.
- If *trans* = 'T',  $A^T$  is used.
- If *trans* = 'C',  $A^H$  is used.
- Scope: **global**
- Specified as: a single character, where:
- For PDSYRK, it must be 'N', 'T', or 'C'.
- For PZSYRK, it must be 'N' or 'T'.
- For PZHERK, it must be 'N' or 'C'.
- n* is the order of the global submatrix  $C$  used in the computation, and:
- If *trans* = 'N', it is the number of rows in submatrix  $A$  used in the computation.
- If *trans* = 'T' or 'C', it is the number of columns in submatrix  $A$  used in the computation.
- Scope: **global**
- Specified as: a fullword integer;  $n \geq 0$ .
- k* has the following meaning:
- If *trans* = 'N', it is the number of columns in submatrix  $A$  used in the computation.
- If *trans* = 'T' or 'C', it is the number of rows in submatrix  $A$  used in the computation.
- Scope: **global**
- Specified as: a fullword integer;  $k \geq 0$ .
- alpha* is the scalar  $\alpha$ .
- Scope: **global**
- Specified as: a number of the data type indicated in Table 63 on page 301.
- a* is the local part of the global general matrix  $A$ . This identifies the **first element** of the local array  $A$ . This subroutine computes the location of the first element of the local subarray used, based on *ia*, *ja*, *desc\_a*, *p*, *q*, *myrow*, and *mycol*; therefore:
- If *trans* = 'N', the leading  $\text{LOCp}(ia+n-1)$  by  $\text{LOCq}(ja+k-1)$  part of the local array  $A$  must contain the local pieces of the leading  $ia+n-1$  by  $ja+k-1$  part of the global matrix.
  - If *trans* = 'T' or 'C', the leading  $\text{LOCp}(ia+k-1)$  by  $\text{LOCq}(ja+n-1)$  part of the local array  $A$  must contain the local pieces of the leading  $ia+k-1$  by  $ja+n-1$  part of the global matrix.
- Note:** No data should be moved to form  $A^T$  or  $A^H$ ; that is, the matrix  $A$  should always be stored in its untransposed form.
- Scope: **local**



Specified as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 63 on page 301. Details about the block-cyclic data distribution of global matrix *A* are stored in *desc\_a*.

*ia* is the row index of the global matrix *A*, identifying the first row of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq M\_A$ , and:

If *trans* = 'N', then  $ia+n-1 \leq M\_A$ .

If *trans* = 'T' or 'C', then  $ia+k-1 \leq M\_A$ .

*ja* is the column index of the global matrix *A*, identifying the first column of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$ , and:

If *trans* = 'N', then  $ja+k-1 \leq N\_A$ .

If *trans* = 'T' or 'C', then  $ja+n-1 \leq N\_A$ .

*desc\_a* is the array descriptor for global matrix *A*, described in the following table:

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $n = 0$ or $k = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	If $n = 0$ or $k = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_A < p$	Global
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < q$	Global
9	LLD_A	The leading dimension of the local array	$LLD\_A \geq \max(1, LOCp(M\_A))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

*beta* is the scalar  $\beta$ .

Scope: **global**

Specified as: a number of the data type indicated in Table 63 on page 301.

## PDSYRK, PZSYRK, and PZHERK

*c* is the local part of the global real or complex symmetric or complex Hermitian matrix *C*. This identifies the **first element** of the local array *C*. This subroutine computes the location of the first element of the local subarray used, based on *ic*, *jc*, *desc\_c*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading LOCp(*ic+n-1*) by LOCq(*jc+n-1*) part of the local array *C* must contain the local pieces of the leading *ic+n-1* by *jc+n-1* part of the global matrix, and:

- If *uplo* = 'U', the leading  $n \times n$  upper triangular part of the global submatrix  $C_{ic:ic+n-1, jc:jc+n-1}$  must contain the upper triangular part of the submatrix, and the strictly lower triangular part is not referenced.
- If *uplo* = 'L', the leading  $n \times n$  lower triangular part of the global submatrix  $C_{ic:ic+n-1, jc:jc+n-1}$  must contain the lower triangular part of the submatrix, and the strictly upper triangular part is not referenced.

When  $\beta$  is zero, *C* need not be set on input.

Scope: **local**

Specified as: an LLD\_C by (at least) LOCq(N\_C) array, containing numbers of the data type indicated in Table 63 on page 301. Details about the block-cyclic data distribution of global matrix *C* are stored in *desc\_c*.

*ic* is the row index of the global matrix *C*, identifying the first row of the submatrix *C*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ic \leq M\_C$  and  $ic+n-1 \leq M\_C$ .

*jc* is the column index of the global matrix *C*, identifying the first column of the submatrix *C*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq jc \leq N\_C$  and  $jc+n-1 \leq N\_C$ .

*desc\_c* is the array descriptor for global matrix *C*, described in the following table:

<i>desc_c</i>	Name	Description	Limits	Scope
1	DTYPE_C	Descriptor type	DTYPE_C=1	Global
2	CTXT_C	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_C	Number of rows in the global matrix	If $n = 0$ : $M\_C \geq 0$ Otherwise: $M\_C \geq 1$	Global
4	N_C	Number of columns in the global matrix	If $n = 0$ : $N\_C \geq 0$ Otherwise: $N\_C \geq 1$	Global
5	MB_C	Row block size	$MB\_C \geq 1$	Global
6	NB_C	Column block size	$NB\_C \geq 1$	Global
7	RSRC_C	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_C < p$	Global

<i>desc_c</i>	Name	Description	Limits	Scope
8	CSRC_C	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq \text{CSRC\_C} < q$	Global
9	LLD_C	The leading dimension of the local array	$\text{LLD\_C} \geq \max(1, \text{LOCp}(\text{M\_C}))$	Local

Specified as: an array of (at least) length 9, containing fullword integers.

### On Return

*c* is the updated local part of the global real or complex symmetric or complex Hermitian matrix *C*, containing the results of the computation.

Scope: **local**

Returned as: an LLD\_C by (at least) LOCq(N\_C) array, containing numbers of the data type indicated in Table 63 on page 301.

### Notes and Coding Rules

- These subroutines accept lowercase letters for the *uplo* and *trans* arguments.
- For PDSYRK, if you specify 'C' for the *trans* argument, it is interpreted as though you specified 'T'.
- The imaginary parts of the diagonal elements of a complex Hermitian matrix *C* are assumed to be zero, so you do not have to set these values. On output, they are set to zero, except when  $\beta$  is one and  $\alpha$  or  $k$  is zero, in which case no computation is performed.
- The matrices must have no common elements; otherwise, results are unpredictable.
- The NUMROC utility subroutine can be used to determine the values of LOCp(M\_) and LOCq(N\_) used in the argument descriptions above. For details, see "Determining the Number of Rows and Columns in Your Local Arrays" on page 28 and "NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process" on page 868.
- For suggested block sizes, see "Coding Tips for Optimizing Parallel Performance" on page 77.
- The following values must be equal: CTXT\_A = CTXT\_C.
- If *C* is **not** contained within a single block, that is:
  - $n + \text{mod}(ic-1, \text{MB\_C}) > \text{MB\_C}$
  - $n + \text{mod}(jc-1, \text{NB\_C}) > \text{NB\_C}$
 then:
  - The global matrix *C* must be distributed using a square block-cyclic distribution; that is, MB\_C = NB\_C.
  - The global matrix *C* must be aligned on a block boundary, that is:
    - $ic-1$  must be a multiple of MB\_C.
    - $jc-1$  must be a multiple of NB\_C.
- If *trans* = 'N':
  - If *C* is **not** contained within a single block, then:
    - The following block sizes must be equal: MB\_A = NB\_C.
    - The global matrix *A* must be aligned on a block row boundary; that is,  $ia-1$  must be a multiple of MB\_A.

## PDSYRK, PZSYRK, and PZHERK

- In the process grid, the process row containing the first row of the submatrix *C* must also contain the first row of the submatrix *A*; that is,  $icrow = iarow$ , where:
  - $icrow = \text{mod}(((ic-1)/MB\_C)+RSRC\_C), p)$
  - $iarow = \text{mod}(((ia-1)/MB\_A)+RSRC\_A), p)$
- 10. If *trans* = 'T' or 'C':
  - If *C* is **not** contained within a single block, then:
    - The following block sizes must be equal:  $NB\_A = MB\_C$ .
    - The global matrix *A* must be aligned on a block column boundary; that is,  $ja-1$  must be a multiple of  $NB\_A$ .
  - In the process grid, the process column containing the first column of the submatrix *C* must also contain the first column of the submatrix *A*; that is,  $iccol = iacol$ , where:
    - $iccol = \text{mod}(((jc-1)/NB\_C)+CSRC\_C), q)$
    - $iacol = \text{mod}(((ja-1)/NB\_A)+CSRC\_A), q)$
- 11. If *C* is contained within a single block:
  - If *trans* = 'N', *A* must be a block row matrix; that is, if  $p > 1$ :
    - $n+\text{mod}(ia-1, MB\_A) \leq MB\_A$
  - If *trans* = 'T' or 'C', *A* must be a block column matrix; that is, if  $q > 1$ :
    - $n+\text{mod}(ja-1, NB\_A) \leq NB\_A$

## Error Conditions

### Computational Errors

None

### Resource Errors

Unable to allocate work space

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. *DTYPE\_A* is invalid.
2. *DTYPE\_C* is invalid.

#### Stage 2:

1. *CTXT\_A* is invalid.

#### Stage 3:

1. This subroutine was called from outside the process grid.

#### Stage 4:

1. *uplo*  $\neq$  'U' or 'L'
2. *trans*  $\neq$ 
  - 'N', 'T', or 'C' for PDSYRK
  - 'N' or 'T' for PZSYRK
  - 'N' or 'C' for PZHERK
3.  $n < 0$  and *trans* = 'N'
4.  $n < 0$  and *trans* = 'T' or 'C'
5.  $n < 0$  and *trans* is invalid.
6.  $k < 0$  and *trans* = 'N'
7.  $k < 0$  and *trans* = 'T' or 'C'
8.  $k < 0$  and *trans* is invalid.
9.  $M\_A < 0$  and ( $n = 0$  or  $k = 0$ );  $M\_A < 1$  otherwise

10.  $N\_A < 0$  and  $(n = 0 \text{ or } k = 0)$ ;  $N\_A < 1$  otherwise
11.  $MB\_A < 1$
12.  $NB\_A < 1$
13.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
14.  $CSRC\_A < 0$  or  $CSRC\_A \geq q$
15.  $ia < 1$
16.  $ja < 1$
17.  $M\_C < 0$  and  $n = 0$ ;  $M\_C < 1$  otherwise
18.  $N\_C < 0$  and  $n = 0$ ;  $N\_C < 1$  otherwise
19.  $MB\_C < 1$
20.  $NB\_C < 1$
21.  $RSRC\_C < 0$  or  $RSRC\_C \geq p$
22.  $CSRC\_C < 0$  or  $CSRC\_C \geq q$
23.  $ic < 1$
24.  $jc < 1$
25.  $CTXT\_A \neq CTXT\_C$

If  $n \neq 0$  and  $k \neq 0$ :

1.  $ia > M\_A$
2.  $ja > N\_A$
3.  $trans = 'N'$  and  $ia+n-1 > M\_A$
4.  $trans = 'N'$  and  $ja+k-1 > N\_A$
5.  $trans = 'T'$  or  $'C'$  and  $ia+k-1 > M\_A$
6.  $trans = 'T'$  or  $'C'$  and  $ja+n-1 > N\_A$

If  $n \neq 0$ :

1.  $ic > M\_C$
2.  $jc > N\_C$
3.  $ic+n-1 > M\_C$
4.  $jc+n-1 > N\_C$

#### Stage 5:

1. If  $C$  is **not** contained within a single block, that is:
  - $n+\text{mod}(ic-1, MB\_C) > MB\_C$
  - $n+\text{mod}(jc-1, NB\_C) > NB\_C$
 and  $NB\_C \neq MB\_C$ .
2.  $trans = 'N'$  and  $NB\_C \neq MB\_A$ .
3.  $trans = 'T'$  or  $'C'$  and  $MB\_C \neq NB\_A$ .

If  $C$  is **not** contained within a single block:

1.  $\text{mod}(ic-1, MB\_C) \neq 0$
2.  $\text{mod}(jc-1, NB\_C) \neq 0$
3.  $trans = 'N'$  and  $\text{mod}(ia-1, MB\_A) \neq 0$
4.  $trans = 'T'$  or  $'C'$  and  $\text{mod}(ja-1, NB\_A) \neq 0$

#### Stage 6:

1.  $LLD\_A < \max(1, \text{LOCp}(M\_A))$
2.  $LLD\_C < \max(1, \text{LOCp}(M\_C))$
3. If  $trans = 'N'$ , then (in the process grid) the process row containing the first row of the submatrix  $C$  does not contain the first row of the submatrix  $A$ ; that is,  $icrow \neq iarow$ , where:
  - $icrow = \text{mod}((((ic-1)/MB\_C)+RSRC\_C), p)$
  - $iarow = \text{mod}((((ia-1)/MB\_A)+RSRC\_A), p)$
4. If  $trans = 'T'$  or  $'C'$ , then (in the process grid) the process column containing the first column of the submatrix  $C$  does not contain the first column of the submatrix  $A$ ; that is,  $iccol \neq iacol$ , where:

## PDSYRK, PZSYRK, and PZHERK

- $iccol = \text{mod}(((jc-1)/NB\_C)+CSRC\_C), q)$
- $iacol = \text{mod}(((ja-1)/NB\_A)+CSRC\_A), q)$

If  $C$  is contained within a single block:

1. If  $trans = 'N'$ :
  - $p > 1$  and  $n+\text{mod}(ia-1, MB\_A) > MB\_A$
2. If  $trans = 'T'$  or  $'C'$ :
  - $q > 1$  and  $n+\text{mod}(ja-1, NB\_A) > NB\_A$

## Examples

### Example 1

This example computes  $C = \alpha AA^T + \beta C$  using a  $2 \times 3$  process grid.

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 3
CALL BLACS_GET (0, 0, ICONXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      UPLO  TRANS  N    K    ALPHA    A  IA  JA    DESC_A  BETA
      |    |    |    |    |        |  |  |    |    |
CALL PDSYRK( 'L' , 'N' , 8 , 5 , 1.0D0 , A , 1 , 1 , DESC_A , 1.0D0 ,

      C  IC  JC  DESC_C
      |  |  |  |
      C , 1 , 1 , DESC_C )
```

	Desc_A	Desc_C
DTYPE_	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	8	8
N_	5	8
MB_	2	2
NB_	2	2
RSRC_	0	0
CSRC_	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>

#### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:
 

```
LLD_A = MAX(1, NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_C = MAX(1, NUMROC(M_C, MB_C, MYROW, RSRC_C, NPROW))
```

 In this example,  $LLD\_A = LLD\_C = 4$  on all processes.

Global general  $8 \times 5$  matrix  $A$  with block size  $2 \times 2$ :

```
B,D      0      1      2
0  [ 0.0  8.0 | 16.0 24.0 | 32.0
    1.0  9.0 | 17.0 25.0 | 33.0 ]
```

1	2.0 10.0		18.0 26.0		34.0
	3.0 11.0		19.0 27.0		35.0
2	4.0 12.0		20.0 28.0		36.0
	5.0 13.0		21.0 29.0		37.0
3	6.0 14.0		22.0 30.0		38.0
	7.0 15.0		23.0 31.0		39.0

The following is the  $2 \times 3$  process grid:

B,D	0	1	2
0	P <sub>00</sub>	P <sub>01</sub>	P <sub>02</sub>
2			
1	P <sub>10</sub>	P <sub>11</sub>	P <sub>12</sub>
3			

Local arrays for A:

p,q	0	1	2
0	0.0 8.0	16.0 24.0	32.0
	1.0 9.0	17.0 25.0	33.0
	4.0 12.0	20.0 28.0	36.0
	5.0 13.0	21.0 29.0	37.0
1	2.0 10.0	18.0 26.0	34.0
	3.0 11.0	19.0 27.0	35.0
	6.0 14.0	22.0 30.0	38.0
	7.0 15.0	23.0 31.0	39.0

Global real symmetric matrix C of order 8 block size  $2 \times 2$ :

B,D	0	1	2	3
0	0.0 .	. .	. .	. .
	1.0 8.0	. .	. .	. .
1	2.0 9.0	15.0 .	. .	. .
	3.0 10.0	16.0 21.0	. .	. .
2	4.0 11.0	17.0 22.0	26.0 .	. .
	5.0 12.0	18.0 23.0	27.0 30.0	. .
3	6.0 13.0	19.0 24.0	28.0 31.0	33.0 .
	7.0 14.0	20.0 25.0	29.0 32.0	34.0 35.0

The following is the  $2 \times 3$  process grid:

B,D	0	1	2
0	P <sub>00</sub>	P <sub>01</sub>	P <sub>02</sub>
2			
1	P <sub>10</sub>	P <sub>11</sub>	P <sub>12</sub>
3			

Local arrays for C:

p,q	0	1	2
	0.0 .	. .	. .
	1.0 8.0	. .	. .

## PDSYRK, PZSYRK, and PZHERK

0	4.0	11.0	.	.	17.0	22.0	26.0	.
	5.0	12.0	.	.	18.0	23.0	27.0	30.0
<hr/>								
1	2.0	9.0	.	.	15.0	.	.	.
	3.0	10.0	.	.	16.0	21.0	.	.
	6.0	13.0	33.0	.	19.0	24.0	28.0	31.0
	7.0	14.0	34.0	35.0	20.0	25.0	29.0	32.0

### Output:

Global real symmetric matrix  $C$  of order 8 with block size  $2 \times 2$ :

B,D	0	1	2	3
0	1920.0 . 2001.0 2093.0	. . . .	. . . .	. . . .
1	2082.0 2179.0 2163.0 2265.0	2275.0 . 2366.0 2466.0	. . . .	. . . .
2	2244.0 2351.0 2325.0 2437.0	2457.0 2562.0 2548.0 2658.0	2666.0 . 2767.0 2875.0	. . . .
3	2406.0 2523.0 2487.0 2609.0	2639.0 2754.0 2730.0 2850.0	2868.0 2981.0 2969.0 3087.0	3093.0 . 3204.0 3320.0

The following is the  $2 \times 3$  process grid:

B,D	0	1	2
0	$P_{00}$	$P_{01}$	$P_{02}$
2			
1	$P_{10}$	$P_{11}$	$P_{12}$
3			

Local arrays for  $C$ :

p,q	0	1	2
0	1920.0 . 2001.0 2093.0 2244.0 2351.0 2325.0 2437.0	. . . . 2457.0 2562.0 2548.0 2658.0	. . . . 2666.0 . 2767.0 2875.0
1	2082.0 2179.0 2163.0 2265.0 2406.0 2523.0 2487.0 2609.0	2275.0 . 2366.0 2466.0 2639.0 2754.0 2730.0 2850.0	. . . . 2868.0 2981.0 2969.0 3087.0

### Example 2

This example computes  $C = \alpha AA^T + \beta C$  using a  $2 \times 3$  process grid.



### Call Statements and Input:

```

ORDER = 'R'
NPROW = 2
NPCOL = 3
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      UPLO  TRANS  N    K    ALPHA  A  IA  JA    DESC_A  BETA
      |    |    |    |    |      |  |  |    |      |
CALL PZSYRK( 'U' , 'N' , 3 , 5 , ALPHA , A , 1 , 1 , DESC_A , BETA ,

      C  IC  JC  DESC_C
      |  |  |  |
      C , 1 , 1 , DESC_C )

ALPHA = (1.0, 1.0)

BETA  = (1.0, 1.0)

```

	Desc_A	Desc_C
DTYPE_	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	3	3
N_	5	3
MB_	2	2
NB_	2	2
RSRC_	0	0
CSRC_	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>

### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:

```

LLD_A = MAX(1, NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_C = MAX(1, NUMROC(M_C, MB_C, MYROW, RSRC_C, NPROW))

```

In this example:

```

LLD_A = LLD_C = 2 on P00, P01, and P02
LLD_A = LLD_C = 1 on P10, P11, and P12

```

Global general  $3 \times 5$  matrix *A* with block size  $2 \times 2$ :

$$\begin{array}{c}
 \text{B,D} \qquad \qquad 0 \qquad \qquad \qquad 1 \qquad \qquad \qquad 2 \\
 0 \quad \left[ \begin{array}{cc|cc|c}
 (2.0,0.0) & (3.0,2.0) & (4.0,1.0) & (1.0,7.0) & (0.0,0.0) \\
 (30.,3.0) & (8.0,0.0) & (2.0,5.0) & (2.0,4.0) & (1.0,2.0) \\
 \hline
 (1.0,3.0) & (2.0,1.0) & (6.0,0.0) & (3.0,2.0) & (2.0,2.0)
 \end{array} \right] \\
 1
 \end{array}$$

The following is the  $2 \times 3$  process grid:

## PDSYRK, PZSYRK, and PZHERK

B,D	0	1	2
0	P <sub>00</sub>	P <sub>01</sub>	P <sub>02</sub>
1	P <sub>10</sub>	P <sub>11</sub>	P <sub>12</sub>

Local arrays for *A*:

p,q	0	1	2
0	(2.0,0.0) (3.0,2.0) (3.0,3.0) (8.0,0.0)	(4.0,1.0) (1.0,7.0) (2.0,5.0) (2.0,4.0)	(0.0,0.0) (1.0,2.0)
1	(1.0,3.0) (2.0,1.0)	(6.0,0.0) (3.0,2.0)	(2.0,2.0)

Global complex symmetric matrix *C* of order 3 with block size 2 × 2:

B,D	0	1
0	(2.0,1.0) (1.0,9.0) . (3.0,1.0)	(4.0,5.0) (6.0,7.0)
1	. .	(8.0,1.0)

The following is the 2 × 3 process grid:

B,D	0	1	2
0	P <sub>00</sub>	P <sub>01</sub>	P <sub>02</sub>
1	P <sub>10</sub>	P <sub>11</sub>	P <sub>12</sub>

Local arrays for *C*:

p,q	0	1	2
0	(2.0,1.0) (1.0,9.0) . (3.0,1.0)	(4.0,5.0) (6.0,7.0)	. .
1	. .	(8.0,1.0)	.

**Output:**

Global complex symmetric matrix *C* of order 3 with block size 2 × 2:

B,D	0	1
0	(-57.0, 13.0) (-63.0, 79.0) . (-28.0, 90.0)	(-24.0, 70.0) (-55.0, 103.0)
1	. .	( 13.0, 75.0)

The following is the 2 × 3 process grid:

B,D	0	1	2
0	P <sub>00</sub>	P <sub>01</sub>	P <sub>02</sub>
1	P <sub>10</sub>	P <sub>11</sub>	P <sub>12</sub>

Local arrays for *C*:

p,q	0	1	2
0	(-57.0, 13.0) (-63.0, 79.0) .	(-24.0, 70.0) (-55.0,103.0)	.
1	.	( 13.0, 75.0)	.

### Example 3

This example computes  $C = \alpha A^H A + \beta C$  using a  $3 \times 2$  process grid.

**Note:** On output, the imaginary parts of the diagonal elements of a complex Hermitian matrix are set to zero, except when  $\beta$  is one and  $\alpha$  or  $k$  is zero.

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 3
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      UPLO  TRANS  N    K    ALPHA  A  IA  JA  DESC_A  BETA
      |    |    |    |    |    |  |  |  |    |    |
CALL PZHERK( 'L' , 'C' , 3 , 5 , ALPHA , A , 1 , 1 , DESC_A , BETA ,

      C  IC  JC  DESC_C
      |  |  |  |
      C , 1 , 1 , DESC_C )

ALPHA = 1.0

BETA  = 1.0
```

	Desc_A	Desc_C
DTYPE_	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	5	3
N_	3	3
MB_	2	2
NB_	2	2
RSRC_	0	0
CSRC_	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>

#### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:  
 $LLD\_A = \text{MAX}(1, \text{NUMROC}(M\_A, MB\_A, MYROW, RSRC\_A, NPROW))$   
 $LLD\_C = \text{MAX}(1, \text{NUMROC}(M\_C, MB\_C, MYROW, RSRC\_C, NPROW))$

In this example:

## PDSYRK, PZSYRK, and PZHERK

$LLD\_A = 2$  on  $P_{00}$ ,  $P_{01}$ ,  $P_{10}$ , and  $P_{11}$   
 $LLD\_A = 1$  on  $P_{20}$  and  $P_{21}$   
 $LLD\_C = 2$  on  $P_{00}$  and  $P_{01}$   
 $LLD\_C = 1$  on  $P_{10}$  and  $P_{11}$

Global general  $5 \times 3$  matrix  $A$  with block size  $2 \times 2$ :

B,D	0	1
0	$\begin{bmatrix} (2.0,0.0) & (3.0,2.0) \\ (3.0,3.0) & (8.0,0.0) \end{bmatrix}$	$\begin{bmatrix} (4.0,1.0) \\ (2.0,5.0) \end{bmatrix}$
1	$\begin{bmatrix} (1.0,3.0) & (2.0,1.0) \\ (3.0,3.0) & (8.0,0.0) \end{bmatrix}$	$\begin{bmatrix} (6.0,0.0) \\ (2.0,5.0) \end{bmatrix}$
2	$\begin{bmatrix} (1.0,9.0) & (3.0,0.0) \end{bmatrix}$	$\begin{bmatrix} (6.0,7.0) \end{bmatrix}$

The following is the  $3 \times 2$  process grid:

B,D	0	1
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$
2	$P_{20}$	$P_{21}$

Local arrays for  $A$ :

p,q	0	1
0	$\begin{bmatrix} (2.0,0.0) & (3.0,2.0) \\ (3.0,3.0) & (8.0,0.0) \end{bmatrix}$	$\begin{bmatrix} (4.0,1.0) \\ (2.0,5.0) \end{bmatrix}$
1	$\begin{bmatrix} (1.0,3.0) & (2.0,1.0) \\ (3.0,3.0) & (8.0,0.0) \end{bmatrix}$	$\begin{bmatrix} (6.0,0.0) \\ (2.0,5.0) \end{bmatrix}$
2	$\begin{bmatrix} (1.0,9.0) & (3.0,0.0) \end{bmatrix}$	$\begin{bmatrix} (6.0,7.0) \end{bmatrix}$

Global complex Hermitian matrix  $C$  of order 3 with block size  $2 \times 2$ :

B,D	0	1
0	$\begin{bmatrix} (6.0,0.0) & . \\ (3.0,4.0) & (10.0,0.0) \end{bmatrix}$	$\begin{bmatrix} . \\ . \end{bmatrix}$
1	$\begin{bmatrix} (9.0,1.0) & (12.0,2.0) \end{bmatrix}$	$\begin{bmatrix} (3.0,0.0) \end{bmatrix}$

The following is the  $3 \times 2$  process grid:

B,D	0	1
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$
--	$P_{20}$	$P_{21}$

Local arrays for  $C$ :

p,q	0	1
0	$\begin{bmatrix} (6.0, .) & . \\ (3.0,4.0) & (10.0, .) \end{bmatrix}$	$\begin{bmatrix} . \\ . \end{bmatrix}$

1	(9.0,1.0) (12.0,2.0)	(3.0, . )
2	.	.

**Output:**

Global complex Hermitian matrix  $C$  of order 3 with block size  $2 \times 2$ :

B,D	0	1
0	(138.0, 0.0) . ( 65.0, 80.0) (165.0, 0.0)	. .
1	(134.0, 46.0) ( 88.0,-88.0)	(199.0, 0.0)

The following is the  $3 \times 2$  process grid:

B,D	0	1
0	P <sub>00</sub>	P <sub>01</sub>
1	P <sub>10</sub>	P <sub>11</sub>
--	P <sub>20</sub>	P <sub>21</sub>

Local arrays for  $C$ :

p,q	0	1
0	(138.0, 0.0) . ( 65.0, 80.0) (165.0, 0.0)	. .
1	(134.0, 46.0) ( 88.0,-88.0)	(199.0, 0.0)
2	.	.

## PDSYR2K, PZSYR2K, and PZHER2K — Rank-2K Update of a Real or Complex Symmetric or a Complex Hermitian Matrix

### Purpose

PDSYR2K and PZSYR2K compute one of the following rank-2k updates:

- 1.  $C \leftarrow \alpha AB^T + \alpha BA^T + \beta C$
- 2.  $C \leftarrow \alpha A^T B + \alpha B^T A + \beta C$

PZHER2K computes one of the following rank-2k updates:

$$3. C \leftarrow \alpha AB^H + \overline{\alpha} BA^H + \beta C$$

$$4. C \leftarrow \alpha A^H B + \overline{\alpha} B^H A + \beta C$$

where, in the formulas above:

- $A$  represents the global general submatrix:
  - For  $trans = 'N'$ , it is  $A_{ia:ia+n-1, ja:ja+k-1}$ .
  - For  $trans = 'T'$  or  $'C'$ , it is  $A_{ia:ia+k-1, ja:ja+n-1}$ .
- $B$  represents the global general submatrix:
  - For  $trans = 'N'$ , it is  $B_{ib:ib+n-1, jb:jb+k-1}$ .
  - For  $trans = 'T'$  or  $'C'$ , it is  $B_{ib:ib+k-1, jb:jb+n-1}$ .
- $C$  represents the global submatrix  $C_{ic:ic+n-1, jc:ic+n-1}$ .
- $\alpha$  and  $\beta$  are scalars.

**Note:** No data should be moved to form  $A^T$ ,  $A^H$ ,  $B^T$ , or  $B^H$ ; that is, the  $A$  and  $B$  matrices should always be stored in their untransposed forms.

In the following two cases, no computation is performed and the subroutine returns after doing some parameter checking:

- $n = 0$
- $\beta$  is one, and  $\alpha$  is zero or  $k = 0$ .

See references [15] and [16].

Table 64. Data Types

$A, B, C, \alpha$	$\beta$	Subprogram
Long-precision real	Long-precision real	PDSYR2K
Long-precision complex	Long-precision complex	PZSYR2K
Long-precision complex	Long-precision real	PZHER2K

### Syntax

<b>Fortran</b>	CALL PDSYR2K   PZSYR2K   PZHER2K ( <i>uplo, trans, n, k, alpha, a, ia, ja, desc_a, b, ib, jb, desc_b, beta, c, ic, jc, desc_c</i> )
<b>C and C++</b>	pdsyr2k   pzsyr2k   pzher2k ( <i>uplo, trans, n, k, alpha, a, ia, ja, desc_a, b, ib, jb, desc_b, beta, c, ic, jc, desc_c</i> );

**On Entry**

- uplo* indicates whether the upper or lower triangular part of the global submatrix *C* is referenced, where:
- If *uplo* = 'U', the upper triangular part is referenced.
- If *uplo* = 'L', the lower triangular part is referenced.
- Scope: **global**
- Specified as: a single character; *uplo* = 'U' or 'L'.
- trans* indicates which computation is performed, where:
- If *trans* = 'N', *A* and *B* are used.
- If *trans* = 'T',  $A^T$  and  $B^T$  are used.
- If *trans* = 'C',  $A^H$  and  $B^H$  are used.
- Scope: **global**
- Specified as: a single character, where:
- For PDSYR2K, it must be 'N', 'T', or 'C'.
- For PZSYR2K, it must be 'N' or 'T'.
- For PZHER2K, it must be 'N' or 'C'.
- n* is the order of the global submatrix *C* used in the computation, and:
- If *trans* = 'N', it is the number of rows in submatrices *A* and *B* used in the computation.
- If *trans* = 'T' or 'C', it is the number of columns in submatrices *A* and *B* used in the computation.
- Scope: **global**
- Specified as: a fullword integer;  $n \geq 0$ .
- k* has the following meaning:
- If *trans* = 'N', it is the number of columns in submatrices *A* and *B* used in the computation.
- If *trans* = 'T' or 'C', it is the number of rows in submatrices *A* and *B* used in the computation.
- Scope: **global**
- Specified as: a fullword integer;  $k \geq 0$ .
- alpha* is the scalar  $\alpha$ .
- Scope: **global**
- Specified as: a number of the data type indicated in Table 64 on page 316.
- a* is the local part of the global general matrix *A*. This identifies the **first element** of the local array *A*. This subroutine computes the location of the first element of the local subarray used, based on *ia*, *ja*, *desc\_a*, *p*, *q*, *myrow*, and *mycol*; therefore:
- If *trans* = 'N', the leading  $LOCp(ia+n-1)$  by  $LOCq(ja+k-1)$  part of the local array *A* must contain the local pieces of the leading  $ia+n-1$  by  $ja+k-1$  part of the global matrix.

## PDSYR2K, PZSYR2K, and PZHER2K

- If  $trans = 'T'$  or  $'C'$ , the leading  $LOCp(ia+k-1)$  by  $LOCq(ja+n-1)$  part of the local array  $A$  must contain the local pieces of the leading  $ia+k-1$  by  $ja+n-1$  part of the global matrix.

**Note:** No data should be moved to form  $A^T$  or  $A^H$ ; that is, the matrix  $A$  should always be stored in its untransposed form.

Scope: **local**

Specified as: an  $LLD\_A$  by (at least)  $LOCq(N\_A)$  array, containing numbers of the data type indicated in Table 64 on page 316. Details about the block-cyclic data distribution of global matrix  $A$  are stored in  $desc\_a$ .

$ia$  is the row index of the global matrix  $A$ , identifying the first row of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq M\_A$ , and:

If  $trans = 'N'$ , then  $ia+n-1 \leq M\_A$ .

If  $trans = 'T'$  or  $'C'$ , then  $ia+k-1 \leq M\_A$ .

$ja$  is the column index of the global matrix  $A$ , identifying the first column of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$ , and:

If  $trans = 'N'$ , then  $ja+k-1 \leq N\_A$ .

If  $trans = 'T'$  or  $'C'$ , then  $ja+n-1 \leq N\_A$ .

$desc\_a$  is the array descriptor for global matrix  $A$ , described in the following table:

$desc\_a$	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $n = 0$ or $k = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	If $n = 0$ or $k = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_A < p$	Global
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < q$	Global



<i>desc_a</i>	Name	Description	Limits	Scope
9	LLD_A	The leading dimension of the local array	$LLD\_A \geq \max(1, LOCp(M\_A))$	Local

Specified as: an array of (at least) length 9, containing fullword integers.

*b* is the local part of the global general matrix **B**. This identifies the **first element** of the local array B. This subroutine computes the location of the first element of the local subarray used, based on *ib*, *jb*, *desc\_b*, *p*, *q*, *myrow*, and *mycol*; therefore:

- If *trans* = 'N', the leading  $LOCp(ib+n-1)$  by  $LOCq(jb+k-1)$  part of the local array B must contain the local pieces of the leading  $ib+n-1$  by  $jb+k-1$  part of the global matrix.
- If *trans* = 'T' or 'C', the leading  $LOCp(ib+k-1)$  by  $LOCq(jb+n-1)$  part of the local array B must contain the local pieces of the leading  $ib+k-1$  by  $jb+n-1$  part of the global matrix.

**Note:** No data should be moved to form  $B^T$  or  $B^H$ ; that is, the matrix **B** should always be stored in its untransposed form.

Scope: **local**

Specified as: an LLD\_B by (at least)  $LOCq(N\_B)$  array, containing numbers of the data type indicated in Table 64 on page 316. Details about the block-cyclic data distribution of global matrix **B** are stored in *desc\_b*.

*ib* is the row index of the global matrix **B**, identifying the first row of the submatrix **B**.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ib \leq M\_B$ , and:

If *trans* = 'N', then  $ib+n-1 \leq M\_B$ .

If *trans* = 'T' or 'C', then  $ib+k-1 \leq M\_B$ .

*jb* is the column index of the global matrix **B**, identifying the first column of the submatrix **B**.

Scope: **global**

Specified as: a fullword integer;  $1 \leq jb \leq N\_B$ , and:

If *trans* = 'N', then  $jb+k-1 \leq N\_B$ .

If *trans* = 'T' or 'C', then  $jb+n-1 \leq N\_B$ .

*desc\_b* is the array descriptor for global matrix **B**, described in the following table:

<i>desc_b</i>	Name	Description	Limits	Scope
1	DTYPE_B	Descriptor type	DTYPE_B=1	Global
2	CTXT_B	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_B	Number of rows in the global matrix	If $n = 0$ or $k = 0$ : $M\_B \geq 0$ Otherwise: $M\_B \geq 1$	Global

## PDSYR2K, PZSYR2K, and PZHER2K

<i>desc_b</i>	Name	Description	Limits	Scope
4	N_B	Number of columns in the global matrix	If $n = 0$ or $k = 0$ : $N\_B \geq 0$ Otherwise: $N\_B \geq 1$	Global
5	MB_B	Row block size	$MB\_B \geq 1$	Global
6	NB_B	Column block size	$NB\_B \geq 1$	Global
7	RSRC_B	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_B < p$	Global
8	CSRC_B	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_B < q$	Global
9	LLD_B	The leading dimension of the local array	$LLD\_B \geq \max(1, LOCp(M\_B))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

*beta* is the scalar  $\beta$ .

Scope: **global**

Specified as: a number of the data type indicated in Table 64 on page 316.

*c* is the local part of the global real symmetric, complex symmetric, or complex Hermitian matrix *C*. This identifies the **first element** of the local array *C*. This subroutine computes the location of the first element of the local subarray used, based on *ic*, *jc*, *desc\_c*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading  $LOCp(ic+n-1)$  by  $LOCq(jc+n-1)$  part of the local array *C* must contain the local pieces of the leading  $ic+n-1$  by  $jc+n-1$  part of the global matrix, and:

- If *uplo* = 'U', the leading  $n \times n$  upper triangular part of the global submatrix  $C_{ic:ic+n-1, jc:ic+n-1}$  must contain the upper triangular part of the submatrix, and the strictly lower triangular part is not referenced.
- If *uplo* = 'L', the leading  $n \times n$  lower triangular part of the global submatrix  $C_{ic:ic+n-1, jc:ic+n-1}$  must contain the lower triangular part of the submatrix, and the strictly upper triangular part is not referenced.

When  $\beta$  is zero, *C* need not be set on input.

Scope: **local**

Specified as: an LLD\_C by (at least)  $LOCq(N\_C)$  array, containing numbers of the data type indicated in Table 64 on page 316. Details about the block-cyclic data distribution of global matrix *C* are stored in *desc\_c*.

*ic* is the row index of the global matrix *C*, identifying the first row of the submatrix *C*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ic \leq M\_C$  and  $ic+n-1 \leq M\_C$ .

*jc* is the column index of the global matrix *C*, identifying the first column of the submatrix *C*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq jc \leq N\_C$  and  $jc+n-1 \leq N\_C$ .

*desc\_c* is the array descriptor for global matrix *C*, described in the following table:

<i>desc_c</i>	Name	Description	Limits	Scope
1	DTYPE_C	Descriptor type	DTYPE_C=1	Global
2	CTXT_C	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_C	Number of rows in the global matrix	If $n = 0$ : $M\_C \geq 0$ Otherwise: $M\_C \geq 1$	Global
4	N_C	Number of columns in the global matrix	If $n = 0$ : $N\_C \geq 0$ Otherwise: $N\_C \geq 1$	Global
5	MB_C	Row block size	$MB\_C \geq 1$	Global
6	NB_C	Column block size	$NB\_C \geq 1$	Global
7	RSRC_C	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_C < p$	Global
8	CSRC_C	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_C < q$	Global
9	LLD_C	The leading dimension of the local array	$LLD\_C \geq \max(1, LOCp(M\_C))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

### On Return

*c* is the updated local part of the global real symmetric, complex symmetric, or complex Hermitian matrix *C*, containing the results of the computation.

Scope: **local**

Returned as: an LLD\_C by (at least) LOCq(N\_C) array, containing numbers of the data type indicated in Table 64 on page 316.

### Notes and Coding Rules

1. These subroutines accept lowercase letters for the *uplo* and *trans* arguments.
2. For PDSYR2K, if you specify 'C' for the *trans* argument, it is interpreted as though you specified 'T'.
3. The imaginary parts of the diagonal elements of a complex Hermitian matrix *C* are assumed to be zero, so you do not have to set these values. On output, they are set to zero, except when  $\beta$  is one and  $\alpha$  or  $k$  is zero, in which case no computation is performed.
4. The matrices must have no common elements; otherwise, results are unpredictable.
5. The NUMROC utility subroutine can be used to determine the values of LOCp(M\_) and LOCq(N\_) used in the argument descriptions above. For details, see "Determining the Number of Rows and Columns in Your Local

Arrays” on page 28 and “NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process” on page 868.

6. For suggested block sizes, see “Coding Tips for Optimizing Parallel Performance” on page 77.
7. The following values must be equal:  $CTXT\_A = CTXT\_B = CTXT\_C$ .
8. If  $trans = 'N'$ :
  - In the process grid, the process row containing the first row of the submatrix  $C$  must also contain the first row of the submatrices  $A$  and  $B$ ; that is:
    - $icrow = iarow$
    - $icrow = ibrow$
    - where:
      - $iarow = \text{mod}(\text{mod}(((ia-1)/MB\_A)+RSRC\_A), p)$
      - $ibrow = \text{mod}(\text{mod}(((ib-1)/MB\_B)+RSRC\_B), p)$
      - $icrow = \text{mod}(\text{mod}(((ic-1)/MB\_C)+RSRC\_C), p)$
  - If looping is required—that is, **either** of the following is true:
    - $k+\text{mod}(ja-1, NB\_A) > NB\_A$
    - $k+\text{mod}(jb-1, NB\_B) > NB\_B$
 then the block column offset of  $A$  must be equal to the block column offset of  $B$ ; that is,  $\text{mod}(ja-1, NB\_A) = \text{mod}(jb-1, NB\_B)$ .
9. If  $trans = 'T'$  or  $'C'$ :
  - In the process grid, the process column containing the first column of the submatrix  $C$  must also contain the first column of the submatrices  $A$  and  $B$ ; that is:
    - $iccol = iacol$
    - $iccol = ibcol$
    - where:
      - $iacol = \text{mod}(\text{mod}(((ja-1)/NB\_A)+CSRC\_A), q)$
      - $ibcol = \text{mod}(\text{mod}(((jb-1)/NB\_B)+CSRC\_B), q)$
      - $iccol = \text{mod}(\text{mod}(((jc-1)/NB\_C)+CSRC\_C), q)$
  - If looping is required—that is, **either** of the following is true:
    - $k+\text{mod}(ia-1, MB\_A) > MB\_A$
    - $k+\text{mod}(ib-1, MB\_B) > MB\_B$
 then the block row offset of  $A$  must be equal to the block row offset of  $B$ ; that is,  $\text{mod}(ia-1, MB\_A) = \text{mod}(ib-1, MB\_B)$
10. If all the following are true:
  - $C$  is contained within a single block, that is:
    - $n+\text{mod}(ic-1, MB\_C) \leq MB\_C$
    - $n+\text{mod}(jc-1, NB\_C) \leq NB\_C$
  - If  $trans = 'N'$ , then (in the process grid) the process column containing the first column of the submatrix  $A$  must also contain the first column of the submatrix  $B$ ; that is,  $iacol = ibcol$ , where:
    - $iacol = \text{mod}(\text{mod}(((ja-1)/NB\_A)+CSRC\_A), q)$
    - $ibcol = \text{mod}(\text{mod}(((jb-1)/NB\_B)+CSRC\_B), q)$
  - If  $trans = 'T'$  or  $'C'$ , then (in the process grid) the process row containing the first row of the submatrix  $A$  must also contain the first row of the submatrix  $B$ ; that is,  $iarow = ibrow$ , where:
    - $iarow = \text{mod}(\text{mod}(((ia-1)/MB\_A)+RSRC\_A), p)$
    - $ibrow = \text{mod}(\text{mod}(((ib-1)/MB\_B)+RSRC\_B), p)$
 then you must follow these rules:

- If  $trans = 'N'$ :
    - $A$  and  $B$  must be block row matrices; that is, if  $p > 1$ :
      - $n + \text{mod}(ia-1, MB\_A) \leq MB\_A$
      - $n + \text{mod}(ib-1, MB\_B) \leq MB\_B$
    - If looping is required, the following block sizes must be equal:  $NB\_A = NB\_B$ .
  - If  $trans = 'T'$  or  $'C'$ :
    - $A$  and  $B$  must be block column matrices; that is, if  $q > 1$ :
      - $n + \text{mod}(ja-1, NB\_A) \leq NB\_A$
      - $n + \text{mod}(jb-1, NB\_B) \leq NB\_B$
    - If looping is required, the following block sizes must be equal:  $MB\_A = MB\_B$ .
11. If the following is true:
- $C$  is **not** contained within a single block.
- or if all the following are true:
- $C$  is contained within a single block.
  - If  $trans = 'N'$ , then (in the process grid) the process column containing the first column of the submatrix  $A$  does not contain the first column of the submatrix  $B$ ; that is,  $iacol \neq ibcol$ , where:
    - $iacol = \text{mod}((((ja-1)/NB\_A)+CSRC\_A), q)$
    - $ibcol = \text{mod}((((jb-1)/NB\_B)+CSRC\_B), q)$
  - If  $trans = 'T'$  or  $'C'$ , then (in the process grid) the process row containing the first row of the submatrix  $A$  does not contain the first row of the submatrix  $B$ ; that is,  $iarow \neq ibrow$ , where:
    - $iarow = \text{mod}((((ia-1)/MB\_A)+RSRC\_A), p)$
    - $ibrow = \text{mod}((((ib-1)/MB\_B)+RSRC\_B), p)$
- then you must follow these rules:
- The global symmetric matrix  $C$  must be distributed using a square block-cyclic distribution; that is,  $MB\_C = NB\_C$ .
  - The global symmetric matrix  $C$  must be aligned on a block boundary, that is:
    - $ic-1$  must be a multiple of  $MB\_C$ .
    - $jc-1$  must be a multiple of  $NB\_C$ .
  - If  $trans = 'N'$ :
    - The following block sizes must be equal:
      - $NB\_A = NB\_B$
      - $MB\_A = MB\_B = NB\_C$ .
    - The global matrices  $A$  and  $B$  must be aligned on a block row boundary, that is:
      - $ia-1$  must be a multiple of  $MB\_A$ .
      - $ib-1$  must be a multiple of  $MB\_B$ .
  - If  $trans = 'T'$  or  $'C'$ :
    - The following block sizes must be equal:
      - $MB\_A = MB\_B$
      - $NB\_A = NB\_B = MB\_C$ .
    - The global matrices  $A$  and  $B$  must be aligned on a block column boundary, that is:
      - $ja-1$  must be a multiple of  $NB\_A$ .
      - $jb-1$  must be a multiple of  $NB\_B$ .

## Error Conditions

### Computational Errors

None

### Resource Errors

Unable to allocate work space

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. DTYPE\_A is invalid.
2. DTYPE\_B is invalid.
3. DTYPE\_C is invalid.

#### Stage 2:

1. CTXT\_A is invalid.

#### Stage 3:

1. This subroutine was called from outside the process grid.

#### Stage 4:

1. *uplo*  $\neq$  'U' or 'L'
2. *trans*  $\neq$ 
  - 'N', 'T', or 'C' for PDSYR2K
  - 'N' or 'T' for PZSYR2K
  - 'N' or 'C' for PZHER2K
3.  $n < 0$  and *trans* = 'N';  $n < 0$  and *trans* = 'T' or 'C';  $n < 0$  and *trans* is invalid.
4.  $k < 0$  and *trans* = 'N';  $k < 0$  and *trans* = 'T' or 'C';  $k < 0$  and *trans* is invalid.
5.  $M\_A < 0$  and ( $n = 0$  or  $k = 0$ );  $M\_A < 1$  otherwise
6.  $N\_A < 0$  and ( $n = 0$  or  $k = 0$ );  $N\_A < 1$  otherwise
7.  $MB\_A < 1$
8.  $NB\_A < 1$
9.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
10.  $CSRC\_A < 0$  or  $CSRC\_A \geq q$
11.  $ia < 1$
12.  $ja < 1$
13.  $M\_B < 0$  and ( $n = 0$  or  $k = 0$ );  $M\_B < 1$  otherwise
14.  $N\_B < 0$  and ( $n = 0$  or  $k = 0$ );  $N\_B < 1$  otherwise
15.  $MB\_B < 1$
16.  $NB\_B < 1$
17.  $RSRC\_B < 0$  or  $RSRC\_B \geq p$
18.  $CSRC\_B < 0$  or  $CSRC\_B \geq q$
19.  $ib < 1$
20.  $jb < 1$
21.  $M\_C < 0$  and  $n = 0$ ;  $M\_C < 1$  otherwise
22.  $N\_C < 0$  and  $n = 0$ ;  $N\_C < 1$  otherwise
23.  $MB\_C < 1$
24.  $NB\_C < 1$
25.  $RSRC\_C < 0$  or  $RSRC\_C \geq p$
26.  $CSRC\_C < 0$  or  $CSRC\_C \geq q$
27.  $ic < 1$
28.  $jc < 1$
29.  $CTXT\_A \neq CTXT\_B$
30.  $CTXT\_A \neq CTXT\_C$

**Stage 5:** If  $n \neq 0$  and  $k \neq 0$ :

1.  $ia > M\_A$
2.  $ja > N\_A$
3.  $trans = 'N'$  and  $ia+n-1 > M\_A$
4.  $trans = 'N'$  and  $ja+k-1 > N\_A$
5.  $trans = 'T'$  or  $'C'$  and  $ia+k-1 > M\_A$
6.  $trans = 'T'$  or  $'C'$  and  $ja+n-1 > N\_A$
7.  $ib > M\_B$
8.  $jb > N\_B$
9.  $trans = 'N'$  and  $ib+n-1 > M\_B$
10.  $trans = 'N'$  and  $jb+k-1 > N\_B$
11.  $trans = 'T'$  or  $'C'$  and  $ib+k-1 > M\_B$
12.  $trans = 'T'$  or  $'C'$  and  $jb+n-1 > N\_B$

If  $n \neq 0$ :

13.  $ic > M\_C$
14.  $jc > N\_C$
15.  $ic+n-1 > M\_C$
16.  $jc+n-1 > N\_C$

**Stage 6:** If  $C$  is contained within a single block, that is:

- $n+\text{mod}(ic-1, MB\_C) \leq MB\_C$
- $n+\text{mod}(jc-1, NB\_C) \leq NB\_C$

and:

- If  $trans = 'N'$ , then (in the process grid) the process column containing the first column of the submatrix  $A$  must also contain the first column of the submatrix  $B$ ; that is,  $iacol = ibcol$ , where:
  - $iacol = \text{mod}((((ja-1)/NB\_A)+CSRC\_A), q)$
  - $ibcol = \text{mod}((((jb-1)/NB\_B)+CSRC\_B), q)$
- If  $trans = 'T'$  or  $'C'$ , then (in the process grid) the process row containing the first row of the submatrix  $A$  must also contain the first row of the submatrix  $B$ ; that is,  $iarow = ibrow$ , where:
  - $iarow = \text{mod}((((ia-1)/MB\_A)+RSRC\_A), p)$
  - $ibrow = \text{mod}((((ib-1)/MB\_B)+RSRC\_B), p)$

then:

- If  $trans = 'N'$ :
  1.  $p > 1$  and  $n+\text{mod}(ia-1, MB\_A) > MB\_A$
  2.  $p > 1$  and  $n+\text{mod}(ib-1, MB\_B) > MB\_B$
  3. Looping is required—that is, **either** of the following is true:
    - $k+\text{mod}(ja-1, NB\_A) > NB\_A$
    - $k+\text{mod}(jb-1, NB\_B) > NB\_B$
 and  $NB\_A \neq NB\_B$ .
- If  $trans = 'T'$  or  $'C'$ :
  1.  $q > 1$  and  $n+\text{mod}(ja-1, NB\_A) > NB\_A$
  2.  $q > 1$  and  $n+\text{mod}(jb-1, NB\_B) > NB\_B$
  3. Looping is required—that is, **either** of the following is true:
    - $k+\text{mod}(ia-1, MB\_A) > MB\_A$
    - $k+\text{mod}(ib-1, MB\_B) > MB\_B$
 and  $MB\_A \neq MB\_B$ .

If  $C$  is **not** contained within a single block, or if  $C$  is contained within a single block and:

## PDSYR2K, PZSYR2K, and PZHER2K

- If  $trans = 'N'$ , then (in the process grid) the process column containing the first column of the submatrix  $A$  does not contain the first column of the submatrix  $B$ ; that is,  $iacol \neq ibcol$ , where:
  - $iacol = \text{mod}(((ja-1)/NB\_A)+CSRC\_A), q)$
  - $ibcol = \text{mod}(((jb-1)/NB\_B)+CSRC\_B), q)$
- If  $trans = 'T'$  or  $'C'$ , then (in the process grid) the process row containing the first row of the submatrix  $A$  does not contain the first row of the submatrix  $B$ ; that is,  $iarow \neq ibrow$ , where:
  - $iarow = \text{mod}(((ia-1)/MB\_A)+RSRC\_A), p)$
  - $ibrow = \text{mod}(((ib-1)/MB\_B)+RSRC\_B), p)$

then:

1.  $MB\_C \neq NB\_C$
2.  $\text{mod}(ic-1, MB\_C) \neq 0$
3.  $\text{mod}(jc-1, NB\_C) \neq 0$
- If  $trans = 'N'$ :
4.  $NB\_C \neq MB\_A$
5.  $NB\_C \neq MB\_B$
6.  $NB\_A \neq NB\_B$
7.  $\text{mod}(ia-1, MB\_A) \neq 0$
8.  $\text{mod}(ib-1, MB\_B) \neq 0$
- If  $trans = 'T'$  or  $'C'$ :
9.  $MB\_C \neq NB\_A$
10.  $MB\_C \neq NB\_B$
11.  $MB\_A \neq MB\_B$
12.  $\text{mod}(ja-1, NB\_A) \neq 0$
13.  $\text{mod}(jb-1, NB\_B) \neq 0$

In all cases:

1.  $LLD\_A < \max(1, LOCp(M\_A))$
2.  $LLD\_B < \max(1, LOCp(M\_B))$
3.  $LLD\_C < \max(1, LOCp(M\_C))$
- If  $trans = 'N'$ :
4. Looping is required and  $\text{mod}(ja-1, NB\_A) \neq \text{mod}(jb-1, NB\_B)$ .
5. In the process grid, the process row containing the first row of the submatrix  $C$  does not contain the first row of the submatrix  $A$ ; that is,  $icrow \neq iarow$ , where:
  - $icrow = \text{mod}(((ic-1)/MB\_C)+RSRC\_C), p)$
  - $iarow = \text{mod}(((ia-1)/MB\_A)+RSRC\_A), p)$
6. In the process grid, the process row containing the first row of the submatrix  $C$  does not contain the first row of the submatrix  $B$ ; that is,  $icrow \neq ibrow$ , where:
  - $icrow = \text{mod}(((ic-1)/MB\_C)+RSRC\_C), p)$
  - $ibrow = \text{mod}(((ib-1)/MB\_B)+RSRC\_B), p)$
- If  $trans = 'T'$  or  $'C'$ :
7. Looping is required and  $\text{mod}(ia-1, MB\_A) \neq \text{mod}(ib-1, MB\_B)$ .
8. In the process grid, the process column containing the first column of the submatrix  $C$  does not contain the first column of the submatrix  $A$ ; that is,  $iccol \neq iacol$ , where:
  - $iccol = \text{mod}(((jc-1)/NB\_C)+CSRC\_C), q)$
  - $iacol = \text{mod}(((ja-1)/NB\_A)+CSRC\_A), q)$
9. In the process grid, the process column containing the first column of the submatrix  $C$  does not contain the first column of the submatrix  $B$ ; that is,  $iccol \neq ibcol$ , where:
  - $iccol = \text{mod}(((jc-1)/NB\_C)+CSRC\_C), q)$
  - $ibcol = \text{mod}(((jb-1)/NB\_B)+CSRC\_B), q)$



## Examples

### Example 1

This example computes  $C = \alpha A^T B + \alpha B^T A + \beta C$  using a  $2 \times 2$  process grid.

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      UPLO  TRANS  N    K    ALPHA  A  IA  JA  DESC_A  B  IB  JB
      |      |      |      |      |  |  |  |      |  |  |  |
CALL PDSYR2K( 'U' , 'T' , 9 , 8 , 1.0D0 , A , 1 , 1 , DESC_A , B , 1 , 1 ,

      DESC_B  BETA  C  IC  JC  DESC_C
      |      |      |  |  |  |
      DESC_B , 0.0D0 , C , 1 , 1 , DESC_C )
```

	Desc_A	Desc_B	Desc_C
DTYPE_	1	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	8	8	9
N_	9	9	9
MB_	2	2	4
NB_	4	4	4
RSRC_	0	0	0
CSRC_	0	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>	See below <sup>2</sup>

#### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:

```
LLD_A = MAX(1, NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_B = MAX(1, NUMROC(M_B, MB_B, MYROW, RSRC_B, NPROW))
LLD_C = MAX(1, NUMROC(M_C, MB_C, MYROW, RSRC_C, NPROW))
```

In this example, LLD\_A = LLD\_B = 4 on all processes, LLD\_C = 5 on P<sub>00</sub> and P<sub>01</sub>, and LLD\_C = 4 on P<sub>10</sub> and P<sub>11</sub>.

Global general  $8 \times 9$  matrix *A* with block size  $2 \times 4$ :

B,D	0	1	2
0	$\begin{bmatrix} 0.0 & -1.0 & -1.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 1.0 \\ 1.0 \end{bmatrix}$
1	$\begin{bmatrix} 0.0 & 0.0 & -1.0 & -1.0 \\ 0.0 & 1.0 & 0.0 & -1.0 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 0.0 & 1.0 & 0.0 \\ 1.0 & 1.0 & 0.0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 1.0 \\ 1.0 \end{bmatrix}$
2	$\begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$	$\begin{bmatrix} -1.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 0.0 & 0.0 \end{bmatrix}$	$\begin{bmatrix} 1.0 \\ 1.0 \end{bmatrix}$

## PDSYR2K, PZSYR2K, and PZHER2K

$$3 \left[ \begin{array}{cccc|cccc} 0.0 & 0.0 & -1.0 & 0.0 & -1.0 & 0.0 & 0.0 & 0.0 \\ -1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -1.0 & 0.0 \end{array} \right] \begin{array}{c} 1.0 \\ 1.0 \end{array}$$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>
3		

Local arrays for *A*:

p,q	0	1
0	$\begin{bmatrix} 0.0 & -1.0 & -1.0 & 0.0 & 1.0 \\ 0.0 & 1.0 & 0.0 & 1.0 & 1.0 \\ 1.0 & 0.0 & 0.0 & 0.0 & 1.0 \\ 1.0 & 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 1.0 \\ -1.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 0.0 & 0.0 \end{bmatrix}$
1	$\begin{bmatrix} 0.0 & 0.0 & -1.0 & -1.0 & 1.0 \\ 0.0 & 1.0 & 0.0 & -1.0 & 1.0 \\ 0.0 & 0.0 & -1.0 & 0.0 & 1.0 \\ -1.0 & 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 0.0 & 1.0 & 0.0 \\ 1.0 & 1.0 & 0.0 & 1.0 \\ -1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & -1.0 & 0.0 \end{bmatrix}$

Global general  $8 \times 9$  matrix *B* with block size  $2 \times 4$ :

B,D	0	1	2
0	$\begin{bmatrix} 0.0 & 1.0 & 1.0 & 0.0 \\ 0.0 & -1.0 & 0.0 & -1.0 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & -1.0 & 0.0 & -1.0 \end{bmatrix}$	$\begin{bmatrix} -1.0 \\ -1.0 \end{bmatrix}$
1	$\begin{bmatrix} 0.0 & 0.0 & 1.0 & 1.0 \\ 0.0 & -1.0 & 0.0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 0.0 & -1.0 & 0.0 \\ -1.0 & -1.0 & 0.0 & -1.0 \end{bmatrix}$	$\begin{bmatrix} -1.0 \\ -1.0 \end{bmatrix}$
2	$\begin{bmatrix} -1.0 & 0.0 & 0.0 & 0.0 \\ -1.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$	$\begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ -1.0 & -1.0 & 0.0 & 0.0 \end{bmatrix}$	$\begin{bmatrix} -1.0 \\ -1.0 \end{bmatrix}$
3	$\begin{bmatrix} 0.0 & 0.0 & 1.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$	$\begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \end{bmatrix}$	$\begin{bmatrix} -1.0 \\ -1.0 \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>
3		

Local arrays for *B*:

p,q	0	1
0	$\begin{bmatrix} 0.0 & 1.0 & 1.0 & 0.0 & -1.0 \\ 0.0 & -1.0 & 0.0 & -1.0 & -1.0 \\ -1.0 & 0.0 & 0.0 & 0.0 & -1.0 \\ -1.0 & 0.0 & 0.0 & 0.0 & -1.0 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & -1.0 & 0.0 & -1.0 \\ 1.0 & 0.0 & 0.0 & 0.0 \\ -1.0 & -1.0 & 0.0 & 0.0 \end{bmatrix}$
1	$\begin{bmatrix} 0.0 & 0.0 & 1.0 & 1.0 & -1.0 \\ 0.0 & -1.0 & 0.0 & 1.0 & -1.0 \\ 0.0 & 0.0 & 1.0 & 0.0 & -1.0 \\ 1.0 & 0.0 & 0.0 & 0.0 & -1.0 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 0.0 & -1.0 & 0.0 \\ -1.0 & -1.0 & 0.0 & -1.0 \\ 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \end{bmatrix}$

**Output:**

Global real symmetric matrix  $C$  of order 9 with block size  $4 \times 4$ :

B,D	0	1	2
0	$\begin{bmatrix} -6.0 & 0.0 & 0.0 & 0.0 \\ . & -6.0 & -2.0 & 0.0 \\ . & . & -6.0 & -2.0 \\ . & . & . & -6.0 \end{bmatrix}$	$\begin{bmatrix} 0.0 & -2.0 & -2.0 & 0.0 \\ -2.0 & -4.0 & 0.0 & -4.0 \\ -2.0 & 0.0 & 2.0 & 0.0 \\ 2.0 & 0.0 & 2.0 & 0.0 \end{bmatrix}$	$\begin{bmatrix} -2.0 \\ -2.0 \\ 6.0 \\ 2.0 \end{bmatrix}$
1	$\begin{bmatrix} . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \end{bmatrix}$	$\begin{bmatrix} -8.0 & -4.0 & 0.0 & -2.0 \\ . & -6.0 & 0.0 & -4.0 \\ . & . & -4.0 & 0.0 \\ . & . & . & -4.0 \end{bmatrix}$	$\begin{bmatrix} 0.0 \\ -6.0 \\ 0.0 \\ -4.0 \end{bmatrix}$
2	$\begin{bmatrix} . & . & . & . \end{bmatrix}$	$\begin{bmatrix} . & . & . & . \end{bmatrix}$	$\begin{bmatrix} -16.0 \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

Local arrays for  $C$ :

p,q	0	1
0	$\begin{bmatrix} -6.0 & 0.0 & 0.0 & 0.0 & -2.0 \\ . & -6.0 & -2.0 & 0.0 & -2.0 \\ . & . & -6.0 & -2.0 & 6.0 \\ . & . & . & -6.0 & 2.0 \\ . & . & . & . & -16.0 \end{bmatrix}$	$\begin{bmatrix} 0.0 & -2.0 & -2.0 & 0.0 \\ -2.0 & -4.0 & 0.0 & -4.0 \\ -2.0 & 0.0 & 2.0 & 0.0 \\ 2.0 & 0.0 & 2.0 & 0.0 \\ . & . & . & . \end{bmatrix}$
1	$\begin{bmatrix} . & . & . & . & 0.0 \\ . & . & . & . & -6.0 \\ . & . & . & . & 0.0 \\ . & . & . & . & -4.0 \end{bmatrix}$	$\begin{bmatrix} -8.0 & -4.0 & 0.0 & -2.0 \\ . & -6.0 & 0.0 & -4.0 \\ . & . & -4.0 & 0.0 \\ . & . & . & -4.0 \end{bmatrix}$

**Example 2**

This example computes  $C = \alpha A^T B + \alpha B^T A + \beta C$  using a  $2 \times 2$  process grid.

## PDSYR2K, PZSYR2K, and PZHER2K

### Call Statements and Input:

```

ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      UPLO  TRANS  N    K    ALPHA  A  IA  JA  DESC_A  B  IB  JB
      |    |    |    |    |      |  |  |  |      |  |  |
CALL PZSYR2K( 'U' , 'T' , 7 , 8 , ALPHA , A , 1 , 1 , DESC_A , B , 1 , 1 ,

      DESC_B  BETA  C  IC  JC  DESC_C
      |      |    |  |  |  |
      DESC_B , BETA , C , 1 , 1 , DESC_C )

ALPHA = (1.0,0.0)

BETA  = (0.0,0.0)

```

	Desc_A	Desc_B	Desc_C
DTYPE_	1	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	8	8	7
N_	7	7	7
MB_	2	2	3
NB_	3	3	3
RSRC_	0	0	0
CSRC_	0	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>	See below <sup>2</sup>

#### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:

```

LLD_A = MAX(1, NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_B = MAX(1, NUMROC(M_B, MB_B, MYROW, RSRC_B, NPROW))
LLD_C = MAX(1, NUMROC(M_C, MB_C, MYROW, RSRC_C, NPROW))

```

In this example, LLD\_A = LLD\_B = 4 on all processes, LLD\_C = 4 on P<sub>00</sub> and P<sub>01</sub>, and LLD\_C = 3 on P<sub>10</sub> and P<sub>11</sub>.

Global general  $8 \times 7$  matrix *A* with block size  $2 \times 3$ :

B,D	0	1	2
0	$\begin{pmatrix} (0.0, 1.0) & (-1.0, 0.0) & (-1.0, 0.0) \\ (0.0, 1.0) & (1.0, 2.0) & (0.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) & (0.0, 1.0) & (0.0, 1.0) \\ (1.0, 2.0) & (0.0, 1.0) & (1.0, 2.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) \\ (0.0, 1.0) \end{pmatrix}$
1	$\begin{pmatrix} (0.0, 1.0) & (0.0, 1.0) & (-1.0, 0.0) \\ (0.0, 1.0) & (1.0, 2.0) & (0.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (-1.0, 0.0) & (0.0, 1.0) & (0.0, 1.0) \\ (-1.0, 0.0) & (1.0, 2.0) & (1.0, 2.0) \end{pmatrix}$	$\begin{pmatrix} (1.0, 2.0) \\ (0.0, 1.0) \end{pmatrix}$
2	$\begin{pmatrix} (1.0, 2.0) & (0.0, 1.0) & (0.0, 1.0) \\ (1.0, 2.0) & (0.0, 1.0) & (0.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) & (-1.0, 0.0) & (0.0, 1.0) \\ (0.0, 1.0) & (1.0, 2.0) & (1.0, 2.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) \\ (0.0, 1.0) \end{pmatrix}$
3	$\begin{pmatrix} (0.0, 1.0) & (0.0, 1.0) & (-1.0, 0.0) \\ (-1.0, 0.0) & (0.0, 1.0) & (0.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) & (-1.0, 0.0) & (0.0, 1.0) \\ (0.0, 1.0) & (0.0, 1.0) & (0.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) \\ (-1.0, 0.0) \end{pmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$
3		

Local arrays for  $A$ :

p,q	0	1
0	$\begin{pmatrix} (0.0, 1.0) & (-1.0, 0.0) & (-1.0, 0.0) & (0.0, 1.0) \\ (0.0, 1.0) & (1.0, 2.0) & (0.0, 1.0) & (0.0, 1.0) \\ (1.0, 2.0) & (0.0, 1.0) & (0.0, 1.0) & (0.0, 1.0) \\ (1.0, 2.0) & (0.0, 1.0) & (0.0, 1.0) & (0.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) & (0.0, 1.0) & (0.0, 1.0) \\ (1.0, 2.0) & (0.0, 1.0) & (1.0, 2.0) \\ (0.0, 1.0) & (-1.0, 0.0) & (0.0, 1.0) \\ (0.0, 1.0) & (1.0, 2.0) & (1.0, 2.0) \end{pmatrix}$
1	$\begin{pmatrix} (0.0, 1.0) & (0.0, 1.0) & (-1.0, 0.0) & (1.0, 2.0) \\ (0.0, 1.0) & (1.0, 2.0) & (0.0, 1.0) & (0.0, 1.0) \\ (0.0, 1.0) & (0.0, 1.0) & (-1.0, 0.0) & (0.0, 1.0) \\ (-1.0, 0.0) & (0.0, 1.0) & (0.0, 1.0) & (-1.0, 0.0) \end{pmatrix}$	$\begin{pmatrix} (-1.0, 0.0) & (0.0, 1.0) & (0.0, 1.0) \\ (-1.0, 0.0) & (1.0, 2.0) & (1.0, 2.0) \\ (0.0, 1.0) & (-1.0, 0.0) & (0.0, 1.0) \\ (0.0, 1.0) & (0.0, 1.0) & (0.0, 1.0) \end{pmatrix}$

Global general  $8 \times 7$  matrix  $B$  with block size  $2 \times 3$ :

B,D	0	1	2
0	$\begin{pmatrix} (0.0, -2.0) & (1.0, -1.0) & (1.0, -1.0) \\ (0.0, -2.0) & (-1.0, -3.0) & (0.0, -2.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, -2.0) & (0.0, -2.0) & (0.0, -2.0) \\ (-1.0, -3.0) & (0.0, -2.0) & (-1.0, -3.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, -2.0) \\ (0.0, -2.0) \end{pmatrix}$
1	$\begin{pmatrix} (0.0, -2.0) & (0.0, -2.0) & (1.0, -1.0) \\ (0.0, -2.0) & (-1.0, -3.0) & (0.0, -2.0) \end{pmatrix}$	$\begin{pmatrix} (1.0, -1.0) & (0.0, -2.0) & (0.0, -2.0) \\ (1.0, -1.0) & (-1.0, -3.0) & (-1.0, -3.0) \end{pmatrix}$	$\begin{pmatrix} (-1.0, -3.0) \\ (0.0, -2.0) \end{pmatrix}$
2	$\begin{pmatrix} (-1.0, -3.0) & (0.0, -2.0) & (0.0, -2.0) \\ (-1.0, -3.0) & (0.0, -2.0) & (0.0, -2.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, -2.0) & (1.0, -1.0) & (0.0, -2.0) \\ (0.0, -2.0) & (-1.0, -3.0) & (-1.0, -3.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, -2.0) \\ (0.0, -2.0) \end{pmatrix}$
3	$\begin{pmatrix} (0.0, -2.0) & (0.0, -2.0) & (1.0, -1.0) \\ (1.0, -1.0) & (0.0, -2.0) & (0.0, -2.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, -2.0) & (1.0, -1.0) & (0.0, -2.0) \\ (0.0, -2.0) & (0.0, -2.0) & (0.0, -2.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, -2.0) \\ (1.0, -1.0) \end{pmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$
3		

## PDSYR2K, PZSYR2K, and PZHER2K

Local arrays for  $B$ :

p,q	0	1
0	$\begin{pmatrix} (0.0, -2.0) & (1.0, -1.0) & (1.0, -1.0) & (0.0, -2.0) \\ (0.0, -2.0) & (-1.0, -3.0) & (0.0, -2.0) & (0.0, -2.0) \\ (-1.0, -3.0) & (0.0, -2.0) & (0.0, -2.0) & (0.0, -2.0) \\ (-1.0, -3.0) & (0.0, -2.0) & (0.0, -2.0) & (0.0, -2.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, -2.0) & (0.0, -2.0) & (0.0, -2.0) \\ (-1.0, -3.0) & (0.0, -2.0) & (-1.0, -3.0) \\ (0.0, -2.0) & (1.0, -1.0) & (0.0, -2.0) \\ (0.0, -2.0) & (-1.0, -3.0) & (-1.0, -3.0) \end{pmatrix}$
1	$\begin{pmatrix} (0.0, -2.0) & (0.0, -2.0) & (1.0, -1.0) & (-1.0, -3.0) \\ (0.0, -2.0) & (-1.0, -3.0) & (0.0, -2.0) & (0.0, -2.0) \\ (0.0, -2.0) & (0.0, -2.0) & (1.0, -1.0) & (0.0, -2.0) \\ (1.0, -1.0) & (0.0, -2.0) & (0.0, -2.0) & (1.0, -1.0) \end{pmatrix}$	$\begin{pmatrix} (1.0, -1.0) & (0.0, -2.0) & (0.0, -2.0) \\ (1.0, -1.0) & (-1.0, -3.0) & (-1.0, -3.0) \\ (0.0, -2.0) & (1.0, -1.0) & (0.0, -2.0) \\ (0.0, -2.0) & (0.0, -2.0) & (0.0, -2.0) \end{pmatrix}$

**Output:**

Global complex symmetric matrix  $C$  of order 7 with block size  $3 \times 3$ :

B,D	0	1	2
0	$\begin{pmatrix} (38.0, -18.0) & (38.0, -6.0) & (26.0, 6.0) \\ . & (38.0, -18.0) & (26.0, 2.0) \\ . & . & (14.0, 6.0) \end{pmatrix}$	$\begin{pmatrix} (32.0, 0.0) & (35.0, -3.0) & (44.0, -16.0) \\ (32.0, 0.0) & (35.0, -7.0) & (44.0, -20.0) \\ (20.0, 8.0) & (23.0, 5.0) & (32.0, 0.0) \end{pmatrix}$	$\begin{pmatrix} (35.0, -7.0) \\ (35.0, -3.0) \\ (23.0, 13.0) \end{pmatrix}$
1	$\begin{pmatrix} . & . & . \\ . & . & . \\ . & . & . \end{pmatrix}$	$\begin{pmatrix} (26.0, -6.0) & (29.0, 7.0) & (38.0, -6.0) \\ . & (32.0, -16.0) & (41.0, -17.0) \\ . & . & (50.0, -30.0) \end{pmatrix}$	$\begin{pmatrix} (29.0, 7.0) \\ (32.0, 0.0) \\ (41.0, -9.0) \end{pmatrix}$
2	$\begin{pmatrix} . & . & . \end{pmatrix}$	$\begin{pmatrix} . & . & . \end{pmatrix}$	$\begin{pmatrix} (32.0, -8.0) \end{pmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

Local arrays for  $C$ :

p,q	0	1
0	$\begin{pmatrix} (38.0, -18.0) & (38.0, -6.0) & (26.0, 6.0) & (35.0, -7.0) \\ . & (38.0, -18.0) & (26.0, 2.0) & (35.0, -3.0) \\ . & . & (14.0, 6.0) & (23.0, 13.0) \\ . & . & . & (32.0, -8.0) \end{pmatrix}$	$\begin{pmatrix} (32.0, 0.0) & (35.0, -3.0) & (44.0, -16.0) \\ (32.0, 0.0) & (35.0, -7.0) & (44.0, -20.0) \\ (20.0, 8.0) & (23.0, 5.0) & (32.0, 0.0) \\ . & . & . \end{pmatrix}$
1	$\begin{pmatrix} . & . & . & (29.0, 7.0) \\ . & . & . & (32.0, 0.0) \\ . & . & . & (41.0, -9.0) \end{pmatrix}$	$\begin{pmatrix} (26.0, -6.0) & (29.0, 7.0) & (38.0, -6.0) \\ . & (32.0, -16.0) & (41.0, -17.0) \\ . & . & (50.0, -30.0) \end{pmatrix}$

### Example 3

This example computes:

$$C \leftarrow \alpha A^H B + \bar{\alpha} B^H A + \beta C$$

using a  $2 \times 2$  process grid.

**Note:** The imaginary parts of the diagonal elements of a complex Hermitian matrix are assumed to be zero, so you do not have to set these values. On output, they are set to zero except when  $\beta$  is one and  $\alpha$  or  $k$  is zero.

### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      UPLO   TRANS   N   K   ALPHA   A   IA   JA   DESC_A   B   IB   JB
      |      |      |   |   |      |   |   |   |      |   |   |
CALL PZHER2K( 'U' , 'C' , 7 , 8 , ALPHA , A , 1 , 1 , DESC_A , B , 1 , 1 ,

      DESC_B   BETA   C   IC   JC   DESC_C
      |      |      |   |   |   |
      DESC_B , BETA , C , 1 , 1 , DESC_C )

ALPHA = (1.0,0.0)

BETA  = 0.0
```

	Desc_A	Desc_B	Desc_C
DTYPE_	1	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	8	8	7
N_	7	7	7
MB_	2	2	3
NB_	3	3	3
RSRC_	0	0	0
CSRC_	0	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>	See below <sup>2</sup>

### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:

```
LLD_A = MAX(1,NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_B = MAX(1,NUMROC(M_B, MB_B, MYROW, RSRC_B, NPROW))
LLD_C = MAX(1,NUMROC(M_C, MB_C, MYROW, RSRC_C, NPROW))
```

In this example, LLD\_A = LLD\_B = 4 on all processes, LLD\_C = 4 on P<sub>00</sub> and P<sub>01</sub>, and LLD\_C = 3 on P<sub>10</sub> and P<sub>11</sub>.

Global general  $8 \times 7$  matrix **A** with block size  $2 \times 3$ :

## PDSYR2K, PZSYR2K, and PZHER2K

B,D	0	1	2
0	$\begin{pmatrix} (0.0, 1.0) & (-1.0, 0.0) & (-1.0, 0.0) \\ (0.0, 1.0) & (1.0, 2.0) & (0.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) & (0.0, 1.0) & (0.0, 1.0) \\ (1.0, 2.0) & (0.0, 1.0) & (1.0, 2.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) \\ (0.0, 1.0) \end{pmatrix}$
1	$\begin{pmatrix} (0.0, 1.0) & (0.0, 1.0) & (-1.0, 0.0) \\ (0.0, 1.0) & (1.0, 2.0) & (0.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (-1.0, 0.0) & (0.0, 1.0) & (0.0, 1.0) \\ (-1.0, 0.0) & (1.0, 2.0) & (1.0, 2.0) \end{pmatrix}$	$\begin{pmatrix} (1.0, 2.0) \\ (0.0, 1.0) \end{pmatrix}$
2	$\begin{pmatrix} (1.0, 2.0) & (0.0, 1.0) & (0.0, 1.0) \\ (1.0, 2.0) & (0.0, 1.0) & (0.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) & (-1.0, 0.0) & (0.0, 1.0) \\ (0.0, 1.0) & (1.0, 2.0) & (1.0, 2.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) \\ (0.0, 1.0) \end{pmatrix}$
3	$\begin{pmatrix} (0.0, 1.0) & (0.0, 1.0) & (-1.0, 0.0) \\ (-1.0, 0.0) & (0.0, 1.0) & (0.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) & (-1.0, 0.0) & (0.0, 1.0) \\ (0.0, 1.0) & (0.0, 1.0) & (0.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) \\ (-1.0, 0.0) \end{pmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$
3		

Local arrays for  $A$ :

p,q	0	1
0	$\begin{pmatrix} (0.0, 1.0) & (-1.0, 0.0) & (-1.0, 0.0) & (0.0, 1.0) \\ (0.0, 1.0) & (1.0, 2.0) & (0.0, 1.0) & (0.0, 1.0) \\ (1.0, 2.0) & (0.0, 1.0) & (0.0, 1.0) & (0.0, 1.0) \\ (1.0, 2.0) & (0.0, 1.0) & (0.0, 1.0) & (0.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) & (0.0, 1.0) & (0.0, 1.0) \\ (1.0, 2.0) & (0.0, 1.0) & (1.0, 2.0) \\ (0.0, 1.0) & (-1.0, 0.0) & (0.0, 1.0) \\ (0.0, 1.0) & (1.0, 2.0) & (1.0, 2.0) \end{pmatrix}$
1	$\begin{pmatrix} (0.0, 1.0) & (0.0, 1.0) & (-1.0, 0.0) & (1.0, 2.0) \\ (0.0, 1.0) & (1.0, 2.0) & (0.0, 1.0) & (0.0, 1.0) \\ (0.0, 1.0) & (0.0, 1.0) & (-1.0, 0.0) & (0.0, 1.0) \\ (-1.0, 0.0) & (0.0, 1.0) & (0.0, 1.0) & (-1.0, 0.0) \end{pmatrix}$	$\begin{pmatrix} (-1.0, 0.0) & (0.0, 1.0) & (0.0, 1.0) \\ (-1.0, 0.0) & (1.0, 2.0) & (1.0, 2.0) \\ (0.0, 1.0) & (-1.0, 0.0) & (0.0, 1.0) \\ (0.0, 1.0) & (0.0, 1.0) & (0.0, 1.0) \end{pmatrix}$

Global general  $8 \times 7$  matrix  $B$  with block size  $2 \times 3$ :

B,D	0	1	2
0	$\begin{pmatrix} (0.0, -2.0) & (1.0, -1.0) & (1.0, -1.0) \\ (0.0, -2.0) & (-1.0, -3.0) & (0.0, -2.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, -2.0) & (0.0, -2.0) & (0.0, -2.0) \\ (-1.0, -3.0) & (0.0, -2.0) & (-1.0, -3.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, -2.0) \\ (0.0, -2.0) \end{pmatrix}$
1	$\begin{pmatrix} (0.0, -2.0) & (0.0, -2.0) & (1.0, -1.0) \\ (0.0, -2.0) & (-1.0, -3.0) & (0.0, -2.0) \end{pmatrix}$	$\begin{pmatrix} (1.0, -1.0) & (0.0, -2.0) & (0.0, -2.0) \\ (1.0, -1.0) & (-1.0, -3.0) & (-1.0, -3.0) \end{pmatrix}$	$\begin{pmatrix} (-1.0, -3.0) \\ (0.0, -2.0) \end{pmatrix}$
2	$\begin{pmatrix} (-1.0, -3.0) & (0.0, -2.0) & (0.0, -2.0) \\ (-1.0, -3.0) & (0.0, -2.0) & (0.0, -2.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, -2.0) & (1.0, -1.0) & (0.0, -2.0) \\ (0.0, -2.0) & (-1.0, -3.0) & (-1.0, -3.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, -2.0) \\ (0.0, -2.0) \end{pmatrix}$
3	$\begin{pmatrix} (0.0, -2.0) & (0.0, -2.0) & (1.0, -1.0) \\ (1.0, -1.0) & (0.0, -2.0) & (0.0, -2.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, -2.0) & (1.0, -1.0) & (0.0, -2.0) \\ (0.0, -2.0) & (0.0, -2.0) & (0.0, -2.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, -2.0) \\ (1.0, -1.0) \end{pmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$
3		



Local arrays for  $B$ :

p,q	0	1
0	$\begin{pmatrix} (0.0, -2.0) & (1.0, -1.0) & (1.0, -1.0) & (0.0, -2.0) \\ (0.0, -2.0) & (-1.0, -3.0) & (0.0, -2.0) & (0.0, -2.0) \\ (-1.0, -3.0) & (0.0, -2.0) & (0.0, -2.0) & (0.0, -2.0) \\ (-1.0, -3.0) & (0.0, -2.0) & (0.0, -2.0) & (0.0, -2.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, -2.0) & (0.0, -2.0) & (0.0, -2.0) \\ (-1.0, -3.0) & (0.0, -2.0) & (-1.0, -3.0) \\ (0.0, -2.0) & (1.0, -1.0) & (0.0, -2.0) \\ (0.0, -2.0) & (-1.0, -3.0) & (-1.0, -3.0) \end{pmatrix}$
1	$\begin{pmatrix} (0.0, -2.0) & (0.0, -2.0) & (1.0, -1.0) & (-1.0, -3.0) \\ (0.0, -2.0) & (-1.0, -3.0) & (0.0, -2.0) & (0.0, -2.0) \\ (0.0, -2.0) & (0.0, -2.0) & (1.0, -1.0) & (0.0, -2.0) \\ (1.0, -1.0) & (0.0, -2.0) & (0.0, -2.0) & (1.0, -1.0) \end{pmatrix}$	$\begin{pmatrix} (1.0, -1.0) & (0.0, -2.0) & (0.0, -2.0) \\ (1.0, -1.0) & (-1.0, -3.0) & (-1.0, -3.0) \\ (0.0, -2.0) & (1.0, -1.0) & (0.0, -2.0) \\ (0.0, -2.0) & (0.0, -2.0) & (0.0, -2.0) \end{pmatrix}$

Output:

Global complex Hermitian matrix  $C$  of order 7 with block size  $3 \times 3$ :

B,D	0	1	2
0	$\begin{pmatrix} (-50.0, 0.0) & (-38.0, 0.0) & (-26.0, -12.0) \\ . & (-50.0, 0.0) & (-30.0, -12.0) \\ . & . & (-26.0, 0.0) \end{pmatrix}$	$\begin{pmatrix} (-32.0, -6.0) & (-35.0, -3.0) & (-48.0, 6.0) \\ (-32.0, -6.0) & (-39.0, -3.0) & (-52.0, 6.0) \\ (-24.0, 6.0) & (-27.0, 9.0) & (-32.0, 18.0) \end{pmatrix}$	$\begin{pmatrix} (-39.0, -3.0) \\ (-35.0, -3.0) \\ (-19.0, 9.0) \end{pmatrix}$
1	$\begin{pmatrix} . & . & . \\ . & . & . \\ . & . & . \end{pmatrix}$	$\begin{pmatrix} (-38.0, 0.0) & (-25.0, 3.0) & (-38.0, 12.0) \\ . & (-48.0, 0.0) & (-49.0, 9.0) \\ . & . & (-62.0, 0.0) \end{pmatrix}$	$\begin{pmatrix} (-25.0, 3.0) \\ (-32.0, 0.0) \\ (-41.0, -9.0) \end{pmatrix}$
2	$\begin{pmatrix} . & . & . \end{pmatrix}$	$\begin{pmatrix} . & . & . \end{pmatrix}$	$\begin{pmatrix} (-40.0, 0.0) \end{pmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

Local arrays for  $C$ :

p,q	0	1
0	$\begin{pmatrix} (-50.0, 0.0) & (-38.0, 0.0) & (-26.0, -12.0) & (-39.0, -3.0) \\ . & (-50.0, 0.0) & (-30.0, -12.0) & (-35.0, -3.0) \\ . & . & (-26.0, 0.0) & (-19.0, 9.0) \\ . & . & . & (-40.0, 0.0) \end{pmatrix}$	$\begin{pmatrix} (-32.0, -6.0) & (-35.0, -3.0) & (-48.0, 6.0) \\ (-32.0, -6.0) & (-39.0, -3.0) & (-52.0, 6.0) \\ (-24.0, 6.0) & (-27.0, 9.0) & (-32.0, 18.0) \\ . & . & . \end{pmatrix}$
1	$\begin{pmatrix} . & . & . & (-25.0, 3.0) \\ . & . & . & (-32.0, 0.0) \\ . & . & . & (-41.0, -9.0) \end{pmatrix}$	$\begin{pmatrix} (-38.0, 0.0) & (-25.0, 3.0) & (-38.0, 12.0) \\ . & (-48.0, 0.0) & (-49.0, 9.0) \\ . & . & (-62.0, 0.0) \end{pmatrix}$

## PDTRAN, PZTRANC, and PZTRANU — Matrix Transpose for a General Matrix

### Purpose

PDTRAN and PZTRANU perform the following matrix computation:

- $C \leftarrow \beta C + \alpha A^T$

PZTRANC performs the following matrix computation:

- $C \leftarrow \beta C + \alpha A^H$

where, in the formula above:

- $A$  represents the global general submatrix  $A_{ia:ia+n-1, ja:ja+m-1}$ .
- $C$  represents the global general submatrix  $C_{ic:ic+m-1, jc:jc+n-1}$ .
- $\alpha$  and  $\beta$  are scalars.

**Note:** No data should be moved to form  $A^T$  or  $A^H$ ; that is, the matrix  $A$  should always be stored in its untransposed form.

In the following three cases, no computation is performed and the subroutine returns after doing some parameter checking:

- $m = 0$
- $n = 0$
- $\alpha$  is zero and  $\beta$  is one.

See references [15] and [16].

Table 65. Data Types

$\alpha, \beta, A, C$	Subprogram
Long-precision real	PDTRAN
Long-precision complex	PZTRANC and PZTRANU

### Syntax

<b>Fortran</b>	CALL PDTRAN   PZTRANC   PZTRANU ( <i>m, n, alpha, a, ia, ja, desc_a, beta, c, ic, jc, desc_c</i> )
<b>C and C++</b>	pdtran   pztranc   pztranu ( <i>m, n, alpha, a, ia, ja, desc_a, beta, c, ic, jc, desc_c</i> );

### On Entry

*m* is the number of rows in submatrix  $C$  and the number of columns in submatrix  $A$  used in the computation.

Scope: **global**

Specified as: a fullword integer;  $m \geq 0$ .

*n* is the number of columns in submatrix  $C$  and the number of rows in submatrix  $A$  used in the computation.

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

*alpha* is the scalar  $\alpha$ .

Scope: **global**

Specified as: a number of the data type indicated in Table 65 on page 336.

- a* is the local part of the global general matrix *A*. This identifies the **first element** of the local array *A*. This subroutine computes the location of the first element of the local subarray used, based on *ia*, *ja*, *desc\_a*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading LOCp(*ia*+*n*−1) by LOCq(*ja*+*m*−1) part of the local array *A* must contain the local pieces of the leading *ia*+*n*−1 by *ja*+*m*−1 part of the global matrix.

**Note:** No data should be moved to form  $A^T$  or  $A^H$ ; that is, the matrix *A* should always be stored in its untransposed form.

Scope: **local**

Specified as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 65 on page 336. Details about the block-cyclic data distribution of global matrix *A* are stored in *desc\_a*.

- ia* is the row index of the global matrix *A*, identifying the first row of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq M\_A$  and  $ia+n-1 \leq M\_A$ .

- ja* is the column index of the global matrix *A*, identifying the first column of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$  and  $ja+m-1 \leq N\_A$ .

- desc\_a* is the array descriptor for global matrix *A*, described in the following table:

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $m = 0$ or $n = 0$ : M_A $\geq 0$ Otherwise: M_A $\geq 1$	Global
4	N_A	Number of columns in the global matrix	If $m = 0$ or $n = 0$ : N_A $\geq 0$ Otherwise: N_A $\geq 1$	Global
5	MB_A	Row block size	MB_A $\geq 1$	Global
6	NB_A	Column block size	NB_A $\geq 1$	Global
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq \text{RSRC\_A} < p$	Global
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq \text{CSRC\_A} < q$	Global
9	LLD_A	The leading dimension of the local array	LLD_A $\geq \max(1, \text{LOCp}(M\_A))$	<b>Local</b>

## PDTRAN, PZTRANC, and PZTRANU

Specified as: an array of (at least) length 9, containing fullword integers.

*beta* is the scalar  $\beta$ .

Scope: **global**

Specified as: a number of the data type indicated in Table 65 on page 336.

*c* is the local part of the global general matrix *C*. This identifies the **first element** of the local array *C*. This subroutine computes the location of the first element of the local subarray used, based on *ic*, *jc*, *desc\_c*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading LOCp(*ic+m-1*) by LOCq(*jc+n-1*) part of the local array *C* must contain the local pieces of the leading *ic+m-1* by *jc+n-1* part of the global matrix.

When  $\beta$  is zero, *C* need not be set on input.

Scope: **local**

Specified as: an LLD\_C by (at least) LOCq(N\_C) array, containing numbers of the data type indicated in Table 65 on page 336. Details about the block-cyclic data distribution of global matrix *C* are stored in *desc\_c*.

*ic* is the row index of the global matrix *C*, identifying the first row of the submatrix *C*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ic \leq M\_C$  and  $ic+m-1 \leq M\_C$ .

*jc* is the column index of the global matrix *C*, identifying the first column of the submatrix *C*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq jc \leq N\_C$  and  $jc+n-1 \leq N\_C$ .

*desc\_c* is the array descriptor for global matrix *C*, described in the following table:

<i>desc_c</i>	Name	Description	Limits	Scope
1	DTYPE_C	Descriptor type	DTYPE_C=1	Global
2	CTXT_C	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_C	Number of rows in the global matrix	If $m = 0$ or $n = 0$ : $M\_C \geq 0$ Otherwise: $M\_C \geq 1$	Global
4	N_C	Number of columns in the global matrix	If $m = 0$ or $n = 0$ : $N\_C \geq 0$ Otherwise: $N\_C \geq 1$	Global
5	MB_C	Row block size	$MB\_C \geq 1$	Global
6	NB_C	Column block size	$NB\_C \geq 1$	Global
7	RSRC_C	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_C < p$	Global
8	CSRC_C	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_C < q$	Global

<i>desc_c</i>	Name	Description	Limits	Scope
9	LLD_C	The leading dimension of the local array	$LLD\_C \geq \max(1, LOCp(M\_C))$	Local

Specified as: an array of (at least) length 9, containing fullword integers.

### On Return

*c* is the updated local part of the global general matrix *C*, containing the results of the computation.

Scope: **local**

Returned as: an LLD\_C by (at least) LOCq(N\_C) array, containing numbers of the data type indicated in Table 65 on page 336.

### Notes and Coding Rules

- The matrices must have no common elements; otherwise, results are unpredictable.
- The NUMROC utility subroutine can be used to determine the values of LOCp(M\_) and LOCq(N\_) used in the argument descriptions above. For details, see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28 and “NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process” on page 868.
- For suggested block sizes, see “Coding Tips for Optimizing Parallel Performance” on page 77.
- The following values must be equal: CTXT\_A = CTXT\_C.
- The coding rules (given in this section) and the error conditions (given in the next section) are written in terms of *adist*. To determine a value for *adist*, check the following conditions, in order, and chose the first value having a true condition:
  - If *A* is a block column matrix, that is:
    - $m + \text{mod}(ja-1, NB\_A) \leq NB\_A$
 then *adist* = ‘C’
  - If *A* is a block row matrix, that is:
    - $n + \text{mod}(ia-1, MB\_A) \leq MB\_A$
 then *adist* = ‘R’
  - If *A* is neither a block column or a block row matrix, then:
    - If  $m \leq n$ , then *adist* = ‘C’.
    - Otherwise, *adist* = ‘R’.
- If *adist* = ‘C’, then you must follow these coding rules:
  - A* must be aligned on a block row boundary, that is:
    - $ia-1$  must be a multiple of MB\_A.
  - C* must be aligned on a block column boundary, that is:
    - $jc-1$  must be a multiple of NB\_C.
  - MB\_A = NB\_C
  - If looping is required—that is, **either** of the following is true:
    - $m + \text{mod}(ja-1, NB\_A) > NB\_A$
    - $m + \text{mod}(ic-1, MB\_C) > MB\_C$
 then:

- The block column offset of *A* must be equal to the block row offset of *C*; that is,  $\text{mod}(ja-1, \text{NB\_A}) = \text{mod}(ic-1, \text{MB\_C})$ .
  - $\text{NB\_A} = \text{MB\_C}$
7. If *adist* = 'R', then you must follow these coding rules:
- *A* must be aligned on a block column boundary, that is:
    - *ja*–1 must be a multiple of  $\text{NB\_A}$ .
  - *C* must be aligned on a block row boundary, that is:
    - *ic*–1 must be a multiple of  $\text{MB\_C}$ .
  - $\text{NB\_A} = \text{MB\_C}$
  - If looping is required—that is, **either** of the following is true:
    - $n + \text{mod}(ia-1, \text{MB\_A}) > \text{MB\_A}$
    - $n + \text{mod}(jc-1, \text{NB\_C}) > \text{NB\_C}$
 then:
    - The block row offset of *A* must be equal to the block column offset of *C*; that is,  $\text{mod}(ia-1, \text{MB\_A}) = \text{mod}(jc-1, \text{NB\_C})$ .
    - $\text{MB\_A} = \text{NB\_C}$

## Error Conditions

### Computational Errors

None

### Resource Errors

Unable to allocate work space

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. *DTYPE\_A* is invalid.
2. *DTYPE\_C* is invalid.

#### Stage 2:

1. *CTXT\_A* is invalid.

#### Stage 3:

1. This subroutine was called from outside the process grid.

#### Stage 4:

1.  $m < 0$
2.  $n < 0$
3.  $M\_A < 0$  and  $(m = 0 \text{ or } n = 0)$ ;  $M\_A < 1$  otherwise
4.  $N\_A < 0$  and  $(m = 0 \text{ or } n = 0)$ ;  $N\_A < 1$  otherwise
5.  $\text{MB\_A} < 1$
6.  $\text{NB\_A} < 1$
7.  $\text{RSRC\_A} < 0$  or  $\text{RSRC\_A} \geq p$
8.  $\text{CSRC\_A} < 0$  or  $\text{CSRC\_A} \geq q$
9.  $ia < 1$
10.  $ja < 1$
11.  $M\_C < 0$  and  $(m = 0 \text{ or } n = 0)$ ;  $M\_C < 1$  otherwise
12.  $N\_C < 0$  and  $(m = 0 \text{ or } n = 0)$ ;  $N\_C < 1$  otherwise
13.  $\text{MB\_C} < 1$
14.  $\text{NB\_C} < 1$
15.  $\text{RSRC\_C} < 0$  or  $\text{RSRC\_C} \geq p$
16.  $\text{CSRC\_C} < 0$  or  $\text{CSRC\_C} \geq q$

17.  $ic < 1$
18.  $jc < 1$
19.  $CTXT\_A \neq CTXT\_C$

#### Stage 5:

**Note:** Some of the following error conditions depend on the value of *adist*—that is, *adist* = 'C' or *adist* = 'R'. For details on determining the value, see “Notes and Coding Rules” on page 339.

If  $m \neq 0$  and  $n \neq 0$ :

1.  $ia > M\_A$
2.  $ja > N\_A$
3.  $ia+n-1 > M\_A$
4.  $ja+m-1 > N\_A$
5.  $ic > M\_C$
6.  $jc > N\_C$
7.  $ic+m-1 > M\_C$
8.  $jc+n-1 > N\_C$

If *adist* = 'C':

1.  $\text{mod}(ia-1, MB\_A) \neq 0$
2.  $\text{mod}(jc-1, NB\_C) \neq 0$
3.  $MB\_A \neq NB\_C$
4. If looping is required—that is, **either** of the following is true:
  - $m+\text{mod}(ja-1, NB\_A) > NB\_A$
  - $m+\text{mod}(ic-1, MB\_C) > MB\_C$
 then:
  - a.  $\text{mod}(ja-1, NB\_A) \neq \text{mod}(ic-1, MB\_C)$
  - b.  $NB\_A \neq MB\_C$ .

If *adist* = 'R':

1.  $\text{mod}(ja-1, NB\_A) \neq 0$
2.  $\text{mod}(ic-1, MB\_C) \neq 0$
3.  $NB\_A \neq MB\_C$
4. If looping is required—that is, **either** of the following is true:
  - $n+\text{mod}(ia-1, MB\_A) > MB\_A$
  - $n+\text{mod}(jc-1, NB\_C) > NB\_C$
 then:
  - a.  $\text{mod}(ia-1, MB\_A) \neq \text{mod}(jc-1, NB\_C)$
  - b.  $MB\_A \neq NB\_C$ .

#### Stage 6:

1.  $LLD\_A < \max(1, \text{LOCp}(M\_A))$
2.  $LLD\_C < \max(1, \text{LOCp}(M\_C))$

## Examples

### Example 1

This example computes  $C = \beta C + \alpha A^T$  using a  $2 \times 2$  process grid.

## PDTRAN, PZTRANC, and PZTRANU

### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      M      N      ALPHA      A      IA      JA      DESC_A      BETA      C      IC      JC      DESC_C
CALL PDTRAN( 9 , 8 , 1.0D0 , A , 1 , 1 , DESC_A , 1.0D0 , C , 1 , 1 , DESC_C )
```

	Desc_A	Desc_C
DTYPE_	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	8	9
N_	9	8
MB_	2	4
NB_	4	2
RSRC_	0	0
CSRC_	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>

### Notes:

- icontxt* is the output of the BLACS\_GRIDINIT call.
- Each process should set the LLD\_ as follows:  
 $LLD\_A = \text{MAX}(1, \text{NUMROC}(M\_A, MB\_A, MYROW, RSRC\_A, NPROW))$   
 $LLD\_C = \text{MAX}(1, \text{NUMROC}(M\_C, MB\_C, MYROW, RSRC\_C, NPROW))$   
 In this example,  $LLD\_A = 4$  on all processes,  $LLD\_C = 5$  on  $P_{00}$  and  $P_{01}$ , and  $LLD\_C = 4$  on  $P_{10}$  and  $P_{11}$ .

Global general  $8 \times 9$  matrix *A* with block size  $2 \times 4$ :

B,D	0	1	2
0	$\begin{bmatrix} 0.0 & -1.0 & -1.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 1.0 \\ 1.0 \end{bmatrix}$
1	$\begin{bmatrix} 0.0 & 0.0 & -1.0 & -1.0 \\ 0.0 & 1.0 & 0.0 & -1.0 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 0.0 & 1.0 & 0.0 \\ 1.0 & 1.0 & 0.0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 1.0 \\ 1.0 \end{bmatrix}$
2	$\begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$	$\begin{bmatrix} -1.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 0.0 & 0.0 \end{bmatrix}$	$\begin{bmatrix} 1.0 \\ 1.0 \end{bmatrix}$
3	$\begin{bmatrix} 0.0 & 0.0 & -1.0 & 0.0 \\ -1.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$	$\begin{bmatrix} -1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & -1.0 & 0.0 \end{bmatrix}$	$\begin{bmatrix} 1.0 \\ 1.0 \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$



Local arrays for A:

p,q	0	1
0	0.0 -1.0 -1.0 0.0 1.0	0.0 0.0 0.0 0.0
	0.0 1.0 0.0 1.0 1.0	0.0 1.0 0.0 1.0
	1.0 0.0 0.0 0.0 1.0	-1.0 0.0 0.0 0.0
	1.0 0.0 0.0 0.0 1.0	1.0 1.0 0.0 0.0
1	0.0 0.0 -1.0 -1.0 1.0	0.0 0.0 1.0 0.0
	0.0 1.0 0.0 -1.0 1.0	1.0 1.0 0.0 1.0
	0.0 0.0 -1.0 0.0 1.0	-1.0 0.0 0.0 0.0
	-1.0 0.0 0.0 0.0 1.0	0.0 0.0 -1.0 0.0

Global general  $9 \times 8$  matrix  $C$  with block size  $4 \times 2$ :

B,D	0	1	2	3
0	0.0 1.0 0.0 -1.0 0.0 0.0 0.0 -1.0	1.0 5.0 0.0 -1.0 1.0 1.0 0.0 1.0	6.0 7.0 0.0 -1.0 0.0 0.0 -1.0 -1.0	8.0 9.0 0.0 1.0 -1.0 0.0 0.0 1.0
1	-1.0 2.0 -1.0 3.0 0.0 4.0 1.0 5.0	0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0	1.0 0.0 -1.0 -1.0 1.0 0.0 0.0 0.0	0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0
2	1.0 2.0	3.0 4.0	1.0 1.0	1.0 1.0

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$

Local arrays for C:

p,q	0	1
0	0.0 1.0 6.0 7.0	1.0 5.0 8.0 9.0
	0.0 -1.0 0.0 -1.0	0.0 -1.0 0.0 1.0
	0.0 0.0 0.0 0.0	1.0 1.0 -1.0 0.0
	0.0 -1.0 -1.0 -1.0	0.0 1.0 0.0 1.0
	1.0 2.0 1.0 1.0	3.0 4.0 1.0 1.0
1	-1.0 2.0 1.0 0.0	0.0 0.0 0.0 0.0
	-1.0 3.0 -1.0 -1.0	0.0 0.0 0.0 0.0
	0.0 4.0 1.0 0.0	1.0 0.0 0.0 0.0
	1.0 5.0 0.0 0.0	0.0 0.0 1.0 0.0

**Output:**

Global general  $9 \times 8$  matrix  $C$  with block size  $4 \times 2$ :

B,D	0	1	2	3
0	0.0 1.0 -1.0 0.0 -1.0 0.0 0.0 0.0	1.0 5.0 0.0 0.0 0.0 1.0 -1.0 0.0	7.0 8.0 0.0 -1.0 0.0 0.0 -1.0 -1.0	8.0 8.0 0.0 1.0 -2.0 0.0 0.0 1.0
1	-1.0 2.0 -1.0 4.0	0.0 1.0 0.0 1.0	0.0 1.0 -1.0 0.0	-1.0 0.0 0.0 0.0

## PDTRAN, PZTRANC, and PZTRANU

1	0.0 4.0 1.0 6.0	2.0 0.0 0.0 1.0	1.0 0.0 0.0 0.0	0.0 -1.0 1.0 0.0
2	2.0 3.0	4.0 5.0	2.0 2.0	2.0 2.0

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for C:

p,q	0	1
0	0.0 1.0 7.0 8.0 -1.0 0.0 0.0 -1.0 -1.0 0.0 0.0 0.0 0.0 0.0 -1.0 -1.0 2.0 3.0 2.0 2.0	1.0 5.0 8.0 8.0 0.0 0.0 0.0 1.0 0.0 1.0 -2.0 0.0 -1.0 0.0 0.0 1.0 4.0 5.0 2.0 2.0
1	-1.0 2.0 0.0 1.0 -1.0 4.0 -1.0 0.0 0.0 4.0 1.0 0.0 1.0 6.0 0.0 0.0	0.0 1.0 -1.0 0.0 0.0 1.0 0.0 0.0 2.0 0.0 0.0 -1.0 0.0 1.0 1.0 0.0

### Example 2

This example computes  $C = \beta C + \alpha A^H$  using a  $2 \times 2$  process grid.

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      M      N      ALPHA      A  IA  JA      DESC_A      BETA      C  IC  JC      DESC_C
CALL PZTRANC( 7 , 8 , ALPHA , A , 1 , 1 , DESC_A , BETA , C , 1 , 1 , DESC_C )

ALPHA = (1.0,0.0)

BETA  = (1.0,0.0)
```

	Desc_A	Desc_C
DTYPE_	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	8	7
N_	7	8
MB_	2	3
NB_	3	2
RSRC_	0	0
CSRC_	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:

LLD\_A = MAX(1, NUMROC(M\_A, MB\_A, MYROW, RSRC\_A, NPROW))  
 LLD\_C = MAX(1, NUMROC(M\_C, MB\_C, MYROW, RSRC\_C, NPROW))

In this example, LLD\_A = 4 on all processes, LLD\_C = 4 on P<sub>00</sub> and P<sub>01</sub>, and LLD\_C = 3 on P<sub>10</sub> and P<sub>11</sub>.

Global general  $8 \times 7$  matrix *A* with block size  $2 \times 3$ :

B,D	0	1	2
0	$\begin{pmatrix} (0.0, 1.0) & (-1.0, 0.0) & (-1.0, 0.0) \\ (0.0, 1.0) & (1.0, 2.0) & (0.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) & (0.0, 1.0) & (0.0, 1.0) \\ (1.0, 2.0) & (0.0, 1.0) & (1.0, 2.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) \\ (0.0, 1.0) \end{pmatrix}$
1	$\begin{pmatrix} (0.0, 1.0) & (0.0, 1.0) & (-1.0, 0.0) \\ (0.0, 1.0) & (1.0, 2.0) & (0.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (-1.0, 0.0) & (0.0, 1.0) & (0.0, 1.0) \\ (-1.0, 0.0) & (1.0, 2.0) & (1.0, 2.0) \end{pmatrix}$	$\begin{pmatrix} (1.0, 2.0) \\ (0.0, 1.0) \end{pmatrix}$
2	$\begin{pmatrix} (1.0, 2.0) & (0.0, 1.0) & (0.0, 1.0) \\ (1.0, 2.0) & (0.0, 1.0) & (0.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) & (-1.0, 0.0) & (0.0, 1.0) \\ (0.0, 1.0) & (1.0, 2.0) & (1.0, 2.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) \\ (0.0, 1.0) \end{pmatrix}$
3	$\begin{pmatrix} (0.0, 1.0) & (0.0, 1.0) & (-1.0, 0.0) \\ (-1.0, 0.0) & (0.0, 1.0) & (0.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) & (-1.0, 0.0) & (0.0, 1.0) \\ (0.0, 1.0) & (0.0, 1.0) & (0.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) \\ (-1.0, 0.0) \end{pmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>
3		

Local arrays for *A*:

p,q	0	1
0	$\begin{pmatrix} (0.0, 1.0) & (-1.0, 0.0) & (-1.0, 0.0) & (0.0, 1.0) \\ (0.0, 1.0) & (1.0, 2.0) & (0.0, 1.0) & (0.0, 1.0) \\ (1.0, 2.0) & (0.0, 1.0) & (0.0, 1.0) & (0.0, 1.0) \\ (1.0, 2.0) & (0.0, 1.0) & (0.0, 1.0) & (0.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) & (0.0, 1.0) & (0.0, 1.0) \\ (1.0, 2.0) & (0.0, 1.0) & (1.0, 2.0) \\ (0.0, 1.0) & (-1.0, 0.0) & (0.0, 1.0) \\ (0.0, 1.0) & (1.0, 2.0) & (1.0, 2.0) \end{pmatrix}$
1	$\begin{pmatrix} (0.0, 1.0) & (0.0, 1.0) & (-1.0, 0.0) & (1.0, 2.0) \\ (0.0, 1.0) & (1.0, 2.0) & (0.0, 1.0) & (0.0, 1.0) \\ (0.0, 1.0) & (0.0, 1.0) & (-1.0, 0.0) & (0.0, 1.0) \\ (-1.0, 0.0) & (0.0, 1.0) & (0.0, 1.0) & (-1.0, 0.0) \end{pmatrix}$	$\begin{pmatrix} (-1.0, 0.0) & (0.0, 1.0) & (0.0, 1.0) \\ (-1.0, 0.0) & (1.0, 2.0) & (1.0, 2.0) \\ (0.0, 1.0) & (-1.0, 0.0) & (0.0, 1.0) \\ (0.0, 1.0) & (0.0, 1.0) & (0.0, 1.0) \end{pmatrix}$

Global general  $7 \times 8$  matrix *C* with block size  $3 \times 2$ :

## PDTRAN, PZTRANC, and PZTRANU

B,D	0	1	2	3
0	( 0.0,-2.0) ( 1.0,-1.0) ( 1.0,-1.0) (-1.0,-3.0) ( 2.0, 0.0) ( 0.0,-2.0)	( 1.0,-1.0) ( 5.0, 3.0) ( 0.0,-2.0) (-1.0,-3.0) ( 1.0,-1.0) ( 1.0,-1.0)	( 6.0, 4.0) ( 7.0, 5.0) ( 0.0,-2.0) (-1.0,-3.0) ( 2.0, 0.0) ( 3.0, 1.0)	( 8.0, 6.0) ( 9.0, 7.0) ( 0.0,-2.0) ( 1.0,-1.0) (-1.0,-3.0) ( 0.0,-2.0)
1	( 3.0, 1.0) (-1.0,-3.0) (-1.0,-3.0) ( 2.0, 0.0) (-1.0,-3.0) ( 3.0, 1.0)	( 0.0,-2.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0)	(-1.0,-3.0) (-1.0,-3.0) ( 1.0,-1.0) ( 1.0,-1.0) (-1.0,-3.0) (-1.0,-3.0)	( 3.0, 1.0) ( 1.0,-1.0) ( 2.0, 0.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0)
2	( 5.0, 3.0) ( 4.0, 2.0)	( 1.0,-1.0) ( 0.0,-2.0)	( 1.0,-1.0) ( 0.0,-2.0)	( 0.0,-2.0) ( 0.0,-2.0)

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for C:

p,q	0	1
0	( 0.0,-2.0) ( 1.0,-1.0) ( 6.0, 4.0) ( 7.0, 5.0) ( 1.0,-1.0) (-1.0,-3.0) ( 0.0,-2.0) (-1.0,-3.0) ( 2.0, 0.0) ( 0.0,-2.0) ( 2.0, 0.0) ( 3.0, 1.0) ( 5.0, 3.0) ( 4.0, 2.0) ( 1.0,-1.0) ( 0.0,-2.0)	( 1.0,-1.0) ( 5.0, 3.0) ( 8.0, 6.0) ( 9.0, 7.0) ( 0.0,-2.0) (-1.0,-3.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 1.0,-1.0) ( 1.0,-1.0) (-1.0,-3.0) ( 0.0,-2.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0)
1	( 3.0, 1.0) (-1.0,-3.0) (-1.0,-3.0) (-1.0,-3.0) (-1.0,-3.0) ( 2.0, 0.0) ( 1.0,-1.0) ( 1.0,-1.0) (-1.0,-3.0) ( 3.0, 1.0) (-1.0,-3.0) (-1.0,-3.0)	( 0.0,-2.0) ( 1.0,-1.0) ( 3.0, 1.0) ( 1.0,-1.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 2.0, 0.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0) ( 0.0,-2.0)

**Output:**

Global general  $7 \times 8$  matrix C with block size  $3 \times 2$ :

B,D	0	1	2	3
0	( 0.0,-3.0) ( 1.0,-2.0) ( 0.0,-1.0) ( 0.0,-5.0) ( 1.0, 0.0) ( 0.0,-3.0)	( 1.0,-2.0) ( 5.0, 2.0) ( 0.0,-3.0) ( 0.0,-5.0) ( 0.0,-1.0) ( 1.0,-2.0)	( 7.0, 2.0) ( 8.0, 3.0) ( 0.0,-3.0) (-1.0,-4.0) ( 2.0,-1.0) ( 3.0, 0.0)	( 8.0, 5.0) ( 8.0, 7.0) ( 0.0,-3.0) ( 1.0,-2.0) (-2.0,-3.0) ( 0.0,-3.0)
1	( 3.0, 0.0) ( 0.0,-5.0) (-1.0,-4.0) ( 2.0,-1.0) (-1.0,-4.0) ( 4.0,-1.0)	(-1.0,-2.0) ( 0.0,-1.0) ( 0.0,-3.0) ( 1.0,-4.0) ( 0.0,-3.0) ( 1.0,-4.0)	(-1.0,-4.0) (-1.0,-4.0) ( 0.0,-1.0) ( 2.0,-3.0) (-1.0,-4.0) ( 0.0,-5.0)	( 3.0, 0.0) ( 1.0,-2.0) ( 1.0, 0.0) ( 0.0,-3.0) ( 0.0,-3.0) ( 0.0,-3.0)
2	( 5.0, 2.0) ( 4.0, 1.0)	( 2.0,-3.0) ( 0.0,-3.0)	( 1.0,-2.0) ( 0.0,-3.0)	( 0.0,-3.0) (-1.0,-2.0)

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for C:

p,q	0	1
0	( 0.0,-3.0) ( 1.0,-2.0) ( 7.0, 2.0) ( 8.0, 3.0) ( 0.0,-1.0) ( 0.0,-5.0) ( 0.0,-3.0) (-1.0,-4.0) ( 1.0, 0.0) ( 0.0,-3.0) ( 2.0,-1.0) ( 3.0, 0.0) ( 5.0, 2.0) ( 4.0, 1.0) ( 1.0,-2.0) ( 0.0,-3.0)	( 1.0,-2.0) ( 5.0, 2.0) ( 8.0, 5.0) ( 8.0, 7.0) ( 0.0,-3.0) ( 0.0,-5.0) ( 0.0,-3.0) ( 1.0,-2.0) ( 0.0,-1.0) ( 1.0,-2.0) (-2.0,-3.0) ( 0.0,-3.0) ( 2.0,-3.0) ( 0.0,-3.0) ( 0.0,-3.0) (-1.0,-2.0)
1	( 3.0, 0.0) ( 0.0,-5.0) (-1.0,-4.0) (-1.0,-4.0) (-1.0,-4.0) ( 2.0,-1.0) ( 0.0,-1.0) ( 2.0,-3.0) (-1.0,-4.0) ( 4.0,-1.0) (-1.0,-4.0) ( 0.0,-5.0)	(-1.0,-2.0) ( 0.0,-1.0) ( 3.0, 0.0) ( 1.0,-2.0) ( 0.0,-3.0) ( 1.0,-4.0) ( 1.0, 0.0) ( 0.0,-3.0) ( 0.0,-3.0) ( 1.0,-4.0) ( 0.0,-3.0) ( 0.0,-3.0)

### Example 3

This example computes  $C = \beta C + \alpha A^T$  using a  $2 \times 2$  process grid.

#### Call Statements and Input:

```

ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      M      N      ALPHA      A      IA      JA      DESC_A      BETA      C      IC      JC      DESC_C
      |      |      |      |      |      |      |      |      |      |      |      |
CALL PZTRANU( 7 , 8 , ALPHA , A , 1 , 1 , DESC_A , BETA , C , 1 , 1 , DESC_C )

      ALPHA = (1.0,0.0)
      BETA  = (1.0,0.0)
    
```

	Desc_A	Desc_C
DTYPE_	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	8	7
N_	7	8
MB_	2	3
NB_	3	2
RSRC_	0	0
CSRC_	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>

#### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.

2. Each process should set the LLD\_ as follows:

LLD\_A = MAX(1, NUMROC(M\_A, MB\_A, MYROW, RSRC\_A, NPROW))

LLD\_C = MAX(1, NUMROC(M\_C, MB\_C, MYROW, RSRC\_C, NPROW))

In this example, LLD\_A = 4 on all processes, LLD\_C = 4 on P<sub>00</sub> and P<sub>01</sub>, and LLD\_C = 3 on P<sub>10</sub> and P<sub>11</sub>.

Global general  $8 \times 7$  matrix *A* with block size  $2 \times 3$ :

## PDTRAN, PZTRANC, and PZTRANU

B,D	0	1	2
0	$\begin{pmatrix} (0.0, 1.0) & (-1.0, 0.0) & (-1.0, 0.0) \\ (0.0, 1.0) & (1.0, 2.0) & (0.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) & (0.0, 1.0) & (0.0, 1.0) \\ (1.0, 2.0) & (0.0, 1.0) & (1.0, 2.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) \\ (0.0, 1.0) \end{pmatrix}$
1	$\begin{pmatrix} (0.0, 1.0) & (0.0, 1.0) & (-1.0, 0.0) \\ (0.0, 1.0) & (1.0, 2.0) & (0.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (-1.0, 0.0) & (0.0, 1.0) & (0.0, 1.0) \\ (-1.0, 0.0) & (1.0, 2.0) & (1.0, 2.0) \end{pmatrix}$	$\begin{pmatrix} (1.0, 2.0) \\ (0.0, 1.0) \end{pmatrix}$
2	$\begin{pmatrix} (1.0, 2.0) & (0.0, 1.0) & (0.0, 1.0) \\ (1.0, 2.0) & (0.0, 1.0) & (0.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) & (-1.0, 0.0) & (0.0, 1.0) \\ (0.0, 1.0) & (1.0, 2.0) & (1.0, 2.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) \\ (0.0, 1.0) \end{pmatrix}$
3	$\begin{pmatrix} (0.0, 1.0) & (0.0, 1.0) & (-1.0, 0.0) \\ (-1.0, 0.0) & (0.0, 1.0) & (0.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) & (-1.0, 0.0) & (0.0, 1.0) \\ (0.0, 1.0) & (0.0, 1.0) & (0.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) \\ (-1.0, 0.0) \end{pmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$
3		

Local arrays for A:

p,q	0	1
0	$\begin{pmatrix} (0.0, 1.0) & (-1.0, 0.0) & (-1.0, 0.0) & (0.0, 1.0) \\ (0.0, 1.0) & (1.0, 2.0) & (0.0, 1.0) & (0.0, 1.0) \\ (1.0, 2.0) & (0.0, 1.0) & (0.0, 1.0) & (0.0, 1.0) \\ (1.0, 2.0) & (0.0, 1.0) & (0.0, 1.0) & (0.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, 1.0) & (0.0, 1.0) & (0.0, 1.0) \\ (1.0, 2.0) & (0.0, 1.0) & (1.0, 2.0) \\ (0.0, 1.0) & (-1.0, 0.0) & (0.0, 1.0) \\ (0.0, 1.0) & (1.0, 2.0) & (1.0, 2.0) \end{pmatrix}$
1	$\begin{pmatrix} (0.0, 1.0) & (0.0, 1.0) & (-1.0, 0.0) & (1.0, 2.0) \\ (0.0, 1.0) & (1.0, 2.0) & (0.0, 1.0) & (0.0, 1.0) \\ (0.0, 1.0) & (0.0, 1.0) & (-1.0, 0.0) & (0.0, 1.0) \\ (-1.0, 0.0) & (0.0, 1.0) & (0.0, 1.0) & (-1.0, 0.0) \end{pmatrix}$	$\begin{pmatrix} (-1.0, 0.0) & (0.0, 1.0) & (0.0, 1.0) \\ (-1.0, 0.0) & (1.0, 2.0) & (1.0, 2.0) \\ (0.0, 1.0) & (-1.0, 0.0) & (0.0, 1.0) \\ (0.0, 1.0) & (0.0, 1.0) & (0.0, 1.0) \end{pmatrix}$

Global general  $7 \times 8$  matrix C with block size  $3 \times 2$ :

B,D	0	1	2	3
0	$\begin{pmatrix} (0.0, -2.0) & (1.0, -1.0) \\ (1.0, -1.0) & (-1.0, -3.0) \\ (2.0, 0.0) & (0.0, -2.0) \end{pmatrix}$	$\begin{pmatrix} (1.0, -1.0) & (5.0, 3.0) \\ (0.0, -2.0) & (-1.0, -3.0) \\ (1.0, -1.0) & (1.0, -1.0) \end{pmatrix}$	$\begin{pmatrix} (6.0, 4.0) & (7.0, 5.0) \\ (0.0, -2.0) & (-1.0, -3.0) \\ (2.0, 0.0) & (3.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (8.0, 6.0) & (9.0, 7.0) \\ (0.0, -2.0) & (1.0, -1.0) \\ (-1.0, -3.0) & (0.0, -2.0) \end{pmatrix}$
1	$\begin{pmatrix} (3.0, 1.0) & (-1.0, -3.0) \\ (-1.0, -3.0) & (2.0, 0.0) \\ (-1.0, -3.0) & (3.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, -2.0) & (1.0, -1.0) \\ (0.0, -2.0) & (0.0, -2.0) \\ (0.0, -2.0) & (0.0, -2.0) \end{pmatrix}$	$\begin{pmatrix} (-1.0, -3.0) & (-1.0, -3.0) \\ (1.0, -1.0) & (1.0, -1.0) \\ (-1.0, -3.0) & (-1.0, -3.0) \end{pmatrix}$	$\begin{pmatrix} (3.0, 1.0) & (1.0, -1.0) \\ (2.0, 0.0) & (0.0, -2.0) \\ (0.0, -2.0) & (0.0, -2.0) \end{pmatrix}$
2	$\begin{pmatrix} (5.0, 3.0) & (4.0, 2.0) \end{pmatrix}$	$\begin{pmatrix} (1.0, -1.0) & (0.0, -2.0) \end{pmatrix}$	$\begin{pmatrix} (1.0, -1.0) & (0.0, -2.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, -2.0) & (0.0, -2.0) \end{pmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

Local arrays for C:

p,q	0	1
0	$\begin{pmatrix} (0.0, -2.0) & (1.0, -1.0) & (6.0, 4.0) & (7.0, 5.0) \\ (1.0, -1.0) & (-1.0, -3.0) & (0.0, -2.0) & (-1.0, -3.0) \\ (2.0, 0.0) & (0.0, -2.0) & (2.0, 0.0) & (3.0, 1.0) \\ (5.0, 3.0) & (4.0, 2.0) & (1.0, -1.0) & (0.0, -2.0) \end{pmatrix}$	$\begin{pmatrix} (1.0, -1.0) & (5.0, 3.0) & (8.0, 6.0) & (9.0, 7.0) \\ (0.0, -2.0) & (-1.0, -3.0) & (0.0, -2.0) & (1.0, -1.0) \\ (1.0, -1.0) & (1.0, -1.0) & (-1.0, -3.0) & (0.0, -2.0) \\ (1.0, -1.0) & (0.0, -2.0) & (0.0, -2.0) & (0.0, -2.0) \end{pmatrix}$
1	$\begin{pmatrix} (3.0, 1.0) & (-1.0, -3.0) & (-1.0, -3.0) & (-1.0, -3.0) \\ (-1.0, -3.0) & (2.0, 0.0) & (1.0, -1.0) & (1.0, -1.0) \\ (-1.0, -3.0) & (3.0, 1.0) & (-1.0, -3.0) & (-1.0, -3.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, -2.0) & (1.0, -1.0) & (3.0, 1.0) & (1.0, -1.0) \\ (0.0, -2.0) & (0.0, -2.0) & (2.0, 0.0) & (0.0, -2.0) \\ (0.0, -2.0) & (0.0, -2.0) & (0.0, -2.0) & (0.0, -2.0) \end{pmatrix}$

**Output:**Global general  $7 \times 8$  matrix  $C$  with block size  $3 \times 2$ :

B,D	0	1	2	3
0	$\begin{pmatrix} (0.0, -1.0) & (1.0, 0.0) \\ (0.0, -1.0) & (0.0, -1.0) \\ (1.0, 0.0) & (0.0, -1.0) \end{pmatrix}$	$\begin{pmatrix} (1.0, 0.0) & (5.0, 4.0) \\ (0.0, -1.0) & (0.0, -1.0) \\ (0.0, -1.0) & (1.0, 0.0) \end{pmatrix}$	$\begin{pmatrix} (7.0, 6.0) & (8.0, 7.0) \\ (0.0, -1.0) & (-1.0, -2.0) \\ (2.0, 1.0) & (3.0, 2.0) \end{pmatrix}$	$\begin{pmatrix} (8.0, 7.0) & (8.0, 7.0) \\ (0.0, -1.0) & (1.0, 0.0) \\ (-2.0, -3.0) & (0.0, -1.0) \end{pmatrix}$
1	$\begin{pmatrix} (3.0, 2.0) & (0.0, -1.0) \\ (-1.0, -2.0) & (2.0, 1.0) \\ (-1.0, -2.0) & (4.0, 3.0) \end{pmatrix}$	$\begin{pmatrix} (-1.0, -2.0) & (0.0, -1.0) \\ (0.0, -1.0) & (1.0, 0.0) \\ (0.0, -1.0) & (1.0, 0.0) \end{pmatrix}$	$\begin{pmatrix} (-1.0, -2.0) & (-1.0, -2.0) \\ (0.0, -1.0) & (2.0, 1.0) \\ (-1.0, -2.0) & (0.0, -1.0) \end{pmatrix}$	$\begin{pmatrix} (3.0, 2.0) & (1.0, 0.0) \\ (1.0, 0.0) & (0.0, -1.0) \\ (0.0, -1.0) & (0.0, -1.0) \end{pmatrix}$
2	$\begin{pmatrix} (5.0, 4.0) & (4.0, 3.0) \end{pmatrix}$	$\begin{pmatrix} (2.0, 1.0) & (0.0, -1.0) \end{pmatrix}$	$\begin{pmatrix} (1.0, 0.0) & (0.0, -1.0) \end{pmatrix}$	$\begin{pmatrix} (0.0, -1.0) & (-1.0, -2.0) \end{pmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

Local arrays for  $C$ :

p,q	0	1
0	$\begin{pmatrix} (0.0, -1.0) & (1.0, 0.0) & (7.0, 6.0) & (8.0, 7.0) \\ (0.0, -1.0) & (0.0, -1.0) & (0.0, -1.0) & (-1.0, -2.0) \\ (1.0, 0.0) & (0.0, -1.0) & (2.0, 1.0) & (3.0, 2.0) \\ (5.0, 4.0) & (4.0, 3.0) & (1.0, 0.0) & (0.0, -1.0) \end{pmatrix}$	$\begin{pmatrix} (1.0, 0.0) & (5.0, 4.0) & (8.0, 7.0) & (8.0, 7.0) \\ (0.0, -1.0) & (0.0, -1.0) & (0.0, -1.0) & (1.0, 0.0) \\ (0.0, -1.0) & (1.0, 0.0) & (-2.0, -3.0) & (0.0, -1.0) \\ (2.0, 1.0) & (0.0, -1.0) & (0.0, -1.0) & (-1.0, -2.0) \end{pmatrix}$
1	$\begin{pmatrix} (3.0, 2.0) & (0.0, -1.0) & (-1.0, -2.0) & (-1.0, -2.0) \\ (-1.0, -2.0) & (2.0, 1.0) & (0.0, -1.0) & (2.0, 1.0) \\ (-1.0, -2.0) & (4.0, 3.0) & (-1.0, -2.0) & (0.0, -1.0) \end{pmatrix}$	$\begin{pmatrix} (-1.0, -2.0) & (0.0, -1.0) & (3.0, 2.0) & (1.0, 0.0) \\ (0.0, -1.0) & (1.0, 0.0) & (1.0, 0.0) & (0.0, -1.0) \\ (0.0, -1.0) & (1.0, 0.0) & (0.0, -1.0) & (0.0, -1.0) \end{pmatrix}$





---

## Chapter 8. Linear Algebraic Equations

The linear algebraic equation subroutines are described in this chapter. These subroutines include a subset of the ScaLAPACK subroutines.

**Note:** The dense and banded linear algebraic equation subroutines are designed in accordance with the proposed ScaLAPACK standard. See references [17], [19], [28], and [29]. If these subroutines do not comply with the standard as approved, IBM will consider updating them to do so. If IBM updates these subroutines, the update could require modifications of the calling application program.

---

### Overview of the Dense Linear Algebraic Equation Subroutines

The dense linear algebraic equation subroutines provide:

- Solutions to linear systems of equations for real and complex general matrices, and their transposes, and for positive definite real symmetric and complex Hermitian matrices.
- Least squares solutions to linear systems of equations for real and complex general matrices.
- Condition number of real and complex general matrices and of positive definite real symmetric and complex Hermitian matrices.

*Table 66. List of Dense Linear Algebraic Equation Subroutines*

Descriptive Name	Long-Precision Subroutine	Page
General Matrix Factorization and Solve	PDGESV PZGESV	355
General Matrix Factorization	PDGETRF PZGETRF	370
General Matrix Solve	PDGETRS PZGETRS	381
General Matrix Inverse	PDGETRI PZGETRI	393
Estimate the Reciprocal of the Condition Number of a General Matrix	PDGECON PZGECON	402
General Matrix QR Factorization	PDGEQRF PZGEQRF	411
General Matrix Least Squares Solution	PDGELS PZGELS	421
Positive Definite Real Symmetric or Complex Hermitian Matrix Factorization and Solve	PDPOSV PZPOSV	435
Positive Definite Real Symmetric or Complex Hermitian Matrix Factorization	PDPOTRF PZPOTRF	449
Positive Definite Real Symmetric or Complex Hermitian Matrix Solve	PDPOTRS PZPOTRS	458
Positive Definite Real Symmetric or Complex Hermitian Matrix Inverse	PDPOTRI PZPOTRI	469

Table 66. List of Dense Linear Algebraic Equation Subroutines (continued)

Descriptive Name	Long-Precision Subroutine	Page
Estimation of the Reciprocal of the Condition Number of a Positive Definite Real Symmetric or Complex Hermitian Matrix	PDPOCON PZPOCON	476

## Overview of the Banded Linear Algebraic Equation Subroutines

The banded linear algebraic equation subroutines provide solutions to linear systems of equations for real positive definite symmetric band matrices, real general tridiagonal matrices, diagonally-dominant real general tridiagonal matrices, and real positive definite symmetric tridiagonal matrices.

Table 67. List of Banded Linear Algebraic Equation Subroutines

Descriptive Name	Long- Precision Subroutine	Page
Positive Definite Symmetric Band Matrix Factorization and Solve	PDPBSV	486
Positive Definite Symmetric Band Matrix Factorization	PDPBTRF	498
Positive Definite Symmetric Band Matrix Solve	PDPBTRS	507
General Tridiagonal Matrix Factorization and Solve	PDGTSV	518
General Tridiagonal Matrix Factorization	PDGTTRF	532
General Tridiagonal Matrix Solve	PDGTTRS	548
Diagonally-Dominant General Tridiagonal Matrix Factorization and Solve	PDDTSV	518
Diagonally-Dominant General Tridiagonal Matrix Factorization	PDDTTRF	532
Diagonally-Dominant General Tridiagonal Matrix Solve	PDDTTRS	548
Positive Definite Symmetric Tridiagonal Matrix Factorization and Solve	PDPTSV	565
Positive Definite Symmetric Tridiagonal Matrix Factorization	PDPTTRF	579
Positive Definite Symmetric Tridiagonal Matrix Solve	PDPTTRS	591

## Overview of the Fortran 90 Sparse Linear Algebraic Equation Subroutines

The Fortran 90 sparse linear algebraic equation subroutines provide solutions to linear systems of equations for a real general sparse matrix. The sparse utility subroutines provided in Parallel ESSL must be used in conjunction with the sparse linear algebraic equation subroutines.

Table 68. List of Fortran 90 Sparse Linear Algebraic Equation Subroutines

Descriptive Name	Long-Precision Subroutine	Page
Allocates Space for an Array Descriptor for a General Sparse Matrix	PADALL	607
Allocates Space for a General Sparse Matrix	PSPALL	609
Allocates Space for a Dense Vector	PGEALL	611
Inserts Local Data into a General Sparse Matrix	PSPINS	613
Inserts Local Data into a Dense Vector	PGEINS	617
Assembles a General Sparse Matrix	PSPASB	619
Assembles a Dense Vector	PGEASB	622
Preconditioner for a General Sparse Matrix	PSPGPR	624
Iterative Linear System Solver for a General Sparse Matrix	PSPGIS	627
Deallocates Space for a Dense Vector	PGEFREE	632
Deallocates Space for a General Sparse Matrix	PSPFREE	633
Deallocates Space for an Array Descriptor for a General Sparse Matrix	PADFREE	635

## Overview of the Fortran 77 Sparse Linear Algebraic Equation Subroutines

The Fortran 77 sparse linear algebraic equation subroutines provide solutions to linear systems of equations for a real general sparse matrix. The sparse utility subroutines provided in Parallel ESSL must be used in conjunction with the sparse linear algebraic equation subroutines.

Table 69. List of The Fortran 77 Sparse Linear Algebraic Equation Subroutines

Descriptive Name	Long-Precision Subroutine	Page
Initializes an Array Descriptor for a General Sparse Matrix	PADINIT	643
Initializes a General Sparse Matrix	PDSPINIT	645
Inserts Local Data into a General Sparse Matrix	PDSPINS	647
Inserts Local Data into a Dense Vector	PDGEINS	652
Assembles a General Sparse Matrix	PDSPASB	655
Assembles a Dense Vector	PDGEASB	659
Preconditioner for a General Sparse Matrix	PDSPGPR	661
Iterative Linear System Solver for a General Sparse Matrix	PDSPGIS	664

---

## Dense Linear Algebraic Equation Subroutines

This section contains the dense linear algebraic equation subroutine descriptions.

## PDGESV and PZGESV — General Matrix Factorization and Solve

### Purpose

These subroutines solve the following systems of equations for multiple right-hand sides:

- $AX = B$

In the formula above:

- $A$  represents the global general submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$ .
- $B$  represents the global general submatrix  $B_{ib:ib+n-1, jb:jb+nrhs-1}$  containing the right-hand sides in its columns.
- $X$  represents the global general submatrix  $B_{ib:ib+n-1, jb:jb+nrhs-1}$  containing the solution vectors in its columns.

If  $n$  is 0, no computation is performed and the subroutine returns after doing some parameter checking. If  $n > 0$  and  $nrhs$  is 0, no solutions are computed and the subroutine returns after factoring the matrix.

See references [17], [19], [23], [38], and [39].

Table 70. Data Types

$A, B$	$ipvt$	Subroutine
Long-precision real	Integer	PDGESV
Long-precision complex	Integer	PZGESV

### Syntax

Fortran	CALL PDGESV   PZGESV ( $n, nrhs, a, ia, ja, desc\_a, ipvt, b, ib, jb, desc\_b, info$ )
C and C++	pdgesv   pzgesv ( $n, nrhs, a, ia, ja, desc\_a, ipvt, b, ib, jb, desc\_b, info$ );

### On Entry

$n$  is the order of the submatrix  $A$  and the number of rows in submatrix  $B$ .

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

$nrhs$  is the number of right-hand sides— that is, the number of columns in submatrix  $B$  used in the computation.

Scope: **global**

Specified as: a fullword integer;  $nrhs \geq 0$ .

$a$  is the local part of the global general matrix  $A$ , used in the system of equations. This identifies the **first element** of the local array  $A$ . This subroutine computes the location of the first element of the local subarray used, based on  $ia, ja, desc\_a, p, q, myrow$ , and  $mycol$ ; therefore, the leading  $LOCp(ia+n-1)$  by  $LOCq(ja+n-1)$  part of the local array  $A$  must contain the local pieces of the leading  $ia+n-1$  by  $ja+n-1$  part of the global matrix.

Scope: **local**

Specified as: an  $LLD\_A$  by (at least)  $LOCq(N\_A)$  array, containing numbers of the data type indicated in Table 70. Details about the square block-cyclic data distribution of global matrix  $A$  are stored in  $desc\_a$ .

*ia* is the row index of the global matrix *A*, identifying the first row of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq M\_A$  and  $ia+n-1 \leq M\_A$ .

*ja* is the column index of the global matrix *A*, identifying the first column of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$  and  $ja+n-1 \leq N\_A$ .

*desc\_a* is the array descriptor for global matrix *A*, described in the following table:

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_A < p$	Global
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < q$	Global
9	LLD_A	The leading dimension of the local array	$LLD\_A \geq \max(1, LOCp(M\_A))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

*ipvt* See On Return.

*b* is the local part of the global general matrix *B*, containing the right-hand sides of the system. This identifies the **first element** of the local array *B*. This subroutine computes the location of the first element of the local subarray used, based on *ib*, *jb*, *desc\_b*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading  $LOCp(ib+n-1)$  by  $LOCq(jb+nrhs-1)$  part of the local array *B* must contain the local pieces of the leading  $ib+n-1$  by  $jb+nrhs-1$  part of the global matrix.

Scope: **local**

Specified as: an  $LLD\_B$  by (at least)  $LOCq(N\_B)$  array, containing numbers of the data type indicated in Table 70 on page 355. Details about the block-cyclic data distribution of global matrix *B* are stored in *desc\_b*.

*ib* is the row index of the global matrix *B*, identifying the first row of the submatrix *B*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ib \leq M\_B$  and  $ib+n-1 \leq M\_B$ .

*jb* is the column index of the global matrix *B*, identifying the first column of the submatrix *B*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq jb \leq N\_B$  and  $jb+nrhs-1 \leq N\_B$ .

*desc\_b* is the array descriptor for global matrix *B*, described in the following table:

<i>desc_b</i>	Name	Description	Limits	Scope
1	DTYPE_B	Descriptor type	DTYPE_B=1	Global
2	CTXT_B	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_B	Number of rows in the global matrix	If $n = 0$ or $nrhs = 0$ : $M\_B \geq 0$ Otherwise: $M\_B \geq 1$	Global
4	N_B	Number of columns in the global matrix	If $n = 0$ or $nrhs = 0$ : $N\_B \geq 0$ Otherwise: $N\_B \geq 1$	Global
5	MB_B	Row block size	$MB\_B \geq 1$	Global
6	NB_B	Column block size	$NB\_B \geq 1$	Global
7	RSRC_B	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_B < p$	Global
8	CSRC_B	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_B < q$	Global
9	LLD_B	The leading dimension of the local array	$LLD\_B \geq \max(1, LOCp(M\_B))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

*info* See On Return.

### On Return

*a* is the updated local part of the global matrix *A*, containing the results of the factorization.

Scope: **local**

Returned as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 70 on page 355.

*ipvt* is the local part of the global vector *ipvt*, containing the pivot indices. This identifies the **first element** of the local array IPVT. This subroutine computes the location of the first element of the local subarray used, based

on  $ia$ ,  $desc\_a$ ,  $p$ , and  $myrow$ ; therefore, the leading  $LOCp(ia+m-1)$  part of the local array  $IPVT$  must contain the local pieces of the leading  $ia+m-1$  part of the global vector.

A copy of the vector  $ipvt$ , with a block size of  $MB\_A$  and global index  $ia$ , is returned to each column of the process grid. The process row over which the first row of  $ipvt$  is distributed is  $RSRC\_A$ .

Scope: **local**

Returned as: an array of (at least) length  $LOCp(ia+m-1)$ , containing fullword integers, where  $ia \leq$  (pivoting indices)  $\leq ia+m-1$ . Details about the block-cyclic data distribution of global vector  $ipvt$  are stored in  $desc\_a$ .

$b$  is the updated local part of the global matrix  $B$ , containing the solution vectors.

Scope: **local**

Returned as: an  $LLD\_B$  by (at least)  $LOCq(N\_B)$  array, containing numbers of the data type indicated in Table 70 on page 355.

$info$  has the following meaning:

If  $info = 0$ , global submatrix  $A$  is not singular, and the factorization and solve completed normally.

If  $info > 0$ , global submatrix  $A$  is singular; that is, one or more columns of  $L$  and the corresponding diagonal of  $U$  contain all zeros. All columns of  $L$  are checked.  $info$  is set equal to  $i$ , the first column of  $L$  with a corresponding  $U = 0$  diagonal element, encountered at  $A_{ia+i-1, ja+i-1}$ . The factorization is completed; however, the solution submatrix  $B$  is not computed.

Scope: **global**

Returned as: a fullword integer;  $info \geq 0$ .

## Notes and Coding Rules

1. In your C program, argument  $info$  must be passed by reference.
2. If  $n > 0$  and  $nrhs = 0$ , only the factorization is computed.
3. The matrices and vector must have no common elements; otherwise, results are unpredictable.
4. The way these subroutines handle singularity differs from ScaLAPACK. These subroutines use the  $info$  argument to provide information about the singularity of  $A$ , like ScaLAPACK, but also provide an error message.
5. The NUMROC utility subroutine can be used to determine the values of  $LOCp(M\_)$  and  $LOCq(N\_)$  used in the argument descriptions above. For details, see "Determining the Number of Rows and Columns in Your Local Arrays" on page 28 and "NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process" on page 868.
6. For suggested block sizes, see "Coding Tips for Optimizing Parallel Performance" on page 77.
7. On both input and output, matrices  $A$  and  $B$  conform to ScaLAPACK format.
8. The following values must be equal:  $CTXT\_A = CTXT\_B$ .
9. The global general matrix  $A$  must be distributed using a square block-cyclic distribution; that is,  $MB\_A = NB\_A$ .
10. The following block sizes must be equal:  $MB\_A = MB\_B$ .



11. The global general matrix  $A$  must be aligned on a block row boundary; that is,  $ia-1$  must be a multiple of  $MB\_A$ .
12. The block row offset of  $A$  must be equal to the block column offset of  $A$ ; that is,  $\text{mod}(ia-1, MB\_A) = \text{mod}(ja-1, NB\_A)$ .
13. The block row offset of  $A$  must be equal to the block row offset of  $B$ ; that is,  $\text{mod}(ia-1, MB\_A) = \text{mod}(ib-1, MB\_B)$ .
14. In the process grid, the process row containing the first row of the submatrix  $A$  must also contain the first row of the submatrix  $B$ ; that is,  $iarow = ibrow$ , where:
  - $iarow = \text{mod}((((ia-1)/MB\_A)+RSRC\_A), p)$
  - $ibrow = \text{mod}((((ib-1)/MB\_B)+RSRC\_B), p)$
15. There is no array descriptor for *ipvt*. It is a column-distributed vector with block size  $MB\_A$ , local arrays of dimension  $LOCp(ia+m-1)$  by 1, and global index  $ia$ . A copy of this vector exists on each column of the process grid, and the process row over which the first column of *ipvt* is distributed is  $RSRC\_A$ .
16. For the Parallel ESSL SMP libraries, these subroutines use nonblocking collective communications; therefore, the `MP_SINGLE_THREAD` environment variable must be set to **NO** (which is the default value). See the *IBM Parallel Environment for AIX: MPI Programming Guide* for more information.

## Error Conditions

### Computational Errors

Matrix  $A$  is a singular matrix. For details, see the description of the *info* argument.

### Resource Errors

Unable to allocate work space

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. `DTYPE_A` is invalid.
2. `DTYPE_B` is invalid.

#### Stage 2:

1. `CTXT_A` is invalid.

#### Stage 3:

1. This subroutine was called from outside the process grid.

#### Stage 4:

1.  $n < 0$
2.  $nrhs < 0$
3.  $M\_A < 0$  and  $n = 0$ ;  $M\_A < 1$  otherwise
4.  $N\_A < 0$  and  $n = 0$ ;  $N\_A < 1$  otherwise
5.  $ia < 1$
6.  $ja < 1$
7.  $MB\_A < 1$
8.  $NB\_A < 1$
9.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
10.  $CSRC\_A < 0$  or  $CSRC\_A \geq q$
11.  $M\_B < 0$  and  $(n = 0$  or  $nrhs = 0)$ ;  $M\_B < 1$  otherwise
12.  $N\_B < 0$  and  $(n = 0$  or  $nrhs = 0)$ ;  $N\_B < 1$  otherwise
13.  $ib < 1$
14.  $jb < 1$

15.  $MB\_B < 1$
16.  $NB\_B < 1$
17.  $RSRC\_B < 0$  or  $RSRC\_B \geq p$
18.  $CSRC\_B < 0$  or  $CSRC\_B \geq q$
19.  $CTXT\_A \neq CTXT\_B$

**Stage 5:** If  $n \neq 0$ :

1.  $ia > M\_A$
  2.  $ja > N\_A$
  3.  $ia+n-1 > M\_A$
  4.  $ja+n-1 > N\_A$
- If  $n \neq 0$  and  $nrhs \neq 0$ :
5.  $ib > M\_B$
  6.  $jb > N\_B$
  7.  $ib+n-1 > M\_B$
  8.  $jb+nrhs-1 > N\_B$

In all cases:

9.  $MB\_A \neq NB\_A$
10.  $\text{mod}(ia-1, MB\_A) \neq \text{mod}(ja-1, NB\_A)$
11.  $MB\_B \neq MB\_A$
12.  $\text{mod}(ia-1, MB\_A) \neq \text{mod}(ib-1, MB\_B)$ .
13.  $\text{mod}(ia-1, MB\_A) \neq 0$
14. In the process grid, the process row containing the first row of the submatrix  $A$  does not contain the first row of the submatrix  $B$ ; that is,  $iarow \neq ibrow$ , where:
  - $iarow = \text{mod}(((ia-1)/MB\_A)+RSRC\_A), p)$
  - $ibrow = \text{mod}(((ib-1)/MB\_B)+RSRC\_B), p)$

**Stage 6:**

1.  $LLD\_A < \max(1, \text{LOCp}(M\_A))$
  2.  $LLD\_B < \max(1, \text{LOCp}(M\_B))$
- Each of the following global input arguments are checked to determine whether its value differs from the value specified on process  $P_{00}$ :
3.  $n$  differs.
  4.  $nrhs$  differs.
  5.  $ia$  differs.
  6.  $ja$  differs.
  7.  $DTYPE\_A$  differs.
  8.  $M\_A$  differs.
  9.  $N\_A$  differs.
  10.  $MB\_A$  differs.
  11.  $NB\_A$  differs.
  12.  $RSRC\_A$  differs.
  13.  $CSRC\_A$  differs.
  14.  $ib$  differs.
  15.  $jb$  differs.
  16.  $DTYPE\_B$  differs.
  17.  $M\_B$  differs.
  18.  $N\_B$  differs.
  19.  $MB\_B$  differs.
  20.  $NB\_B$  differs.
  21.  $RSRC\_B$  differs.
  22.  $CSRC\_B$  differs.

## Examples

### Example 1

This example solves the real system  $AX = B$  where  $A$  is a  $9 \times 9$  real general matrix and  $B$  contains 5 right-hand sides using a  $2 \times 2$  process grid. By specifying  $RSRC\_A = 1$ , the rows of global matrix  $A$  and the elements of global vector *ipvt* are distributed over the process grid starting in the second row of the process grid.

This example uses a global submatrix  $B$  within a global matrix  $B$  by specifying  $ib = 1$  and  $jb = 2$ .

By specifying  $RSRC\_B = 1$ , the rows of global matrix  $B$  are distributed over the process grid starting in the second row of the process grid. In addition, by specifying  $CSRC\_B = 1$ , the columns of global matrix  $B$  are distributed over the process grid starting in the second column of the process grid.

### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      N  NRHS  A  IA  JA  DESC_A  IPVT  B  IB  JB  DESC_B  INFO
      |  |    |  |  |  |        |  |  |  |  |  |
CALL PDGESV (9 , 5 , A , 1 , 1 , DESC_A , IPVT , B , 1 , 2 , DESC_B , INFO)
```

	Desc_A	Desc_B
DTYPE_	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	9	9
N_	9	6
MB_	3	3
NB_	3	2
RSRC_	1	1
CSRC_	0	1
LLD_	See below <sup>2</sup>	See below <sup>2</sup>

### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.

2. Each process should set the LLD\_ as follows:

```
LLD_A = MAX(1, NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_B = MAX(1, NUMROC(M_B, MB_B, MYROW, RSRC_B, NPROW))
```

In this example,  $LLD\_A = LLD\_B = 3$  on  $P_{00}$  and  $P_{01}$ , and  $LLD\_A = LLD\_B = 6$  on  $P_{10}$  and  $P_{11}$ .

Global general  $9 \times 9$  matrix  $A$  with block size  $3 \times 3$ :

B,D            0                    1                    2

$$0 \quad \left[ \begin{array}{ccc|ccc|ccc} 1.0 & 1.2 & 1.4 & 1.6 & 1.8 & 2.0 & 2.2 & 2.4 & 2.6 \\ 1.2 & 1.0 & 1.2 & 1.4 & 1.6 & 1.8 & 2.0 & 2.2 & 2.4 \\ 1.4 & 1.2 & 1.0 & 1.2 & 1.4 & 1.6 & 1.8 & 2.0 & 2.2 \end{array} \right]$$

1	-----			-----			-----		
	1.6	1.4	1.2	1.0	1.2	1.4	1.6	1.8	2.0
	1.8	1.6	1.4	1.2	1.0	1.2	1.4	1.6	1.8
2	2.0	1.8	1.6	1.4	1.2	1.0	1.2	1.4	1.6
	-----			-----			-----		
	2.2	2.0	1.8	1.6	1.4	1.2	1.0	1.2	1.4
2	2.4	2.2	2.0	1.8	1.6	1.4	1.2	1.0	1.2
	2.6	2.4	2.2	2.0	1.8	1.6	1.4	1.2	1.0
	-----			-----			-----		

The following is the  $2 \times 2$  process grid:

B,D	02	1
-----		
1	P <sub>00</sub>	P <sub>01</sub>
-----		
0	P <sub>10</sub>	P <sub>11</sub>
2		

**Note:** The first row of  $A$  begins in the second row of the process grid.

Local arrays for  $A$ :

p, q	0						1		
0	1.6	1.4	1.2	1.6	1.8	2.0	1.0	1.2	1.4
	1.8	1.6	1.4	1.4	1.6	1.8	1.2	1.0	1.2
	2.0	1.8	1.6	1.2	1.4	1.6	1.5	1.3	1.0
1	1.0	1.2	1.4	2.2	2.4	2.6	1.6	1.8	2.0
	1.2	1.0	1.2	2.0	2.2	2.4	1.4	1.6	1.8
	1.4	1.2	1.0	1.8	2.0	2.2	1.2	1.4	1.6
	2.2	2.0	1.8	1.0	1.2	1.4	1.6	1.4	1.2
	2.4	2.2	2.0	1.2	1.0	1.2	1.8	1.6	1.4
	2.6	2.4	2.2	1.4	1.2	1.0	2.0	1.8	1.6

After the global matrix  $B$  is distributed over the process grid, only a portion of the global data structure is used—that is, global submatrix  $B$ . Following is the global  $9 \times 5$  submatrix  $B$ , starting at row 1 and column 2 in global general  $9 \times 6$  matrix  $B$  with block size  $3 \times 2$ :

B,D	0	1	2
0	. 93.0	186.0 279.0	372.0 465.0
	. 84.4	168.8 253.2	337.6 422.0
	. 76.6	153.2 229.8	306.4 383.0
1	. 70.0	140.0 210.0	280.0 350.0
	. 65.0	130.0 195.0	260.0 325.0
	. 62.0	124.0 186.0	248.0 310.0
2	. 61.4	122.8 184.2	245.6 307.0
	. 63.6	127.2 190.8	254.4 318.0
	. 69.0	138.0 207.0	276.0 345.0

The following is the  $2 \times 2$  process grid:

B,D	1	0 2
-----		
1	P <sub>00</sub>	P <sub>01</sub>
-----		
0	P <sub>10</sub>	P <sub>11</sub>
2		

**Note:** The first row of  $B$  begins in the second row of the process grid, and the first column of  $B$  begins in the second column of the process grid.

Local arrays for  $B$ :

p,q	0	1
0	140.0 210.0	. 70.0 280.0 350.0
	130.0 195.0	. 65.0 260.0 325.0
	124.0 186.0	. 62.0 248.0 310.0
1	186.0 279.0	. 93.0 372.0 465.0
	168.8 253.2	. 84.4 337.6 422.0
	153.2 229.8	. 76.6 306.4 383.0
	122.8 184.2	. 61.4 245.6 307.0
	127.2 190.8	. 63.6 254.4 318.0
	138.0 207.0	. 69.0 276.0 345.0

**Output:**

Global general  $9 \times 9$  transformed matrix  $A$  with block size  $3 \times 3$ :

B,D	0	1	2
0	2.6 2.4 2.2 0.4 0.3 0.6 0.5 -0.4 0.4	2.0 1.8 1.6 0.8 1.1 1.4 0.8 1.2 1.6	1.4 1.2 1.0 1.7 1.9 2.2 2.0 2.4 2.8
1	0.5 -0.3 0.0 0.6 -0.3 0.0 0.7 -0.2 0.0	0.4 0.8 1.2 0.0 0.4 0.8 0.0 0.0 0.4	1.6 2.0 2.4 1.2 1.6 2.0 0.8 1.2 1.6
2	0.8 -0.2 0.0 0.8 -0.1 0.0 0.9 -0.1 0.0	0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0	0.4 0.8 1.2 0.0 0.4 0.8 0.0 0.0 0.4

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
1	$P_{00}$	$P_{01}$
0 2	$P_{10}$	$P_{11}$

**Note:** The first row of  $A$  begins in the second row of the process grid.

Local arrays for  $A$ :

p,q	0	1
0	0.5 -0.3 0.0 1.6 2.0 2.4 0.6 -0.3 0.0 1.2 1.6 2.0 0.7 -0.2 0.0 0.8 1.2 1.6	0.4 0.8 1.2 0.0 0.4 0.8 0.0 0.0 0.4
1	2.6 2.4 2.2 1.4 1.2 1.0 0.4 0.3 0.6 1.7 1.9 2.2 0.5 -0.4 0.4 2.0 2.4 2.8 0.8 -0.2 0.0 0.4 0.8 1.2 0.8 -0.1 0.0 0.0 0.4 0.8 0.9 -0.1 0.0 0.0 0.0 0.4	2.0 1.8 1.6 0.8 1.1 1.4 0.8 1.2 1.6 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0

Global vector  $ipvt$  of length 9 with block size 3:

B,D	0
	9
0	9
	9
	--
	9
1	9
	9
	--
	9
2	9
	9

**Note:** A copy of *ipvt* is distributed across each column of the process grid.

The following is the  $2 \times 2$  process grid:

B,D		
1	P <sub>00</sub>	P <sub>01</sub>
0	P <sub>10</sub>	P <sub>11</sub>
2		

**Note:** The first row of *ipvt* begins in the second row of the process grid.

Local arrays for *ipvt*:

p,q	0	1
	9	9
0	9	9
	9	9
	--	--
	9	9
	9	9
	9	9
1	9	9
	9	9
	9	9

After the global matrix *B* is distributed over the process grid, only a portion of the global data structure is used—that is, global submatrix *B*. Following is the global  $9 \times 5$  submatrix *B*, starting at row 1 and column 2 in global general  $9 \times 6$  matrix *B* with block size  $3 \times 2$ :

B,D	0	1	2
	. 1.0	2.0 3.0	4.0 5.0
0	. 2.0	4.0 6.0	8.0 10.0
	. 3.0	6.0 9.0	12.0 15.0
	--	--	--
	. 4.0	8.0 12.0	16.0 20.0
1	. 5.0	10.0 15.0	20.0 25.0
	. 6.0	12.0 18.0	24.0 30.0
	--	--	--
	. 7.0	14.0 21.0	28.0 35.0
2	. 8.0	16.0 24.0	32.0 40.0
	. 9.0	18.0 27.0	36.0 45.0

The following is the  $2 \times 2$  process grid:

B,D	1	0 2
1	P <sub>00</sub>	P <sub>01</sub>
0	P <sub>10</sub>	P <sub>11</sub>
2		

**Note:** The first row of  $B$  begins in the second row of the process grid, and the first column of  $B$  begins in the second column of the process grid.

Local arrays for  $B$ :

p,q	0	1
0	8.0 12.0 10.0 15.0 12.0 18.0	. 4.0 16.0 20.0 . 5.0 20.0 25.0 . 6.0 24.0 30.0
1	2.0 3.0 4.0 6.0 6.0 9.0 14.0 21.0 16.0 24.0 18.0 27.0	. 1.0 4.0 5.0 . 2.0 8.0 10.0 . 3.0 12.0 15.0 . 7.0 28.0 35.0 . 8.0 32.0 40.0 . 9.0 36.0 45.0

The value of *info* is 0 on all processes.

### Example 2

This example solves the complex system  $AX = B$  where  $A$  is a  $9 \times 9$  complex general matrix and  $B$  contains 5 right-hand sides using a  $2 \times 2$  process grid. By specifying  $RSRC\_A = 1$ , the rows of global matrix  $A$  and the elements of global vector *ipvt* are distributed over the process grid starting in the second row of the process grid.

This example uses a global submatrix  $B$  within a global matrix  $B$  by specifying  $ib = 1$  and  $jb = 2$ .

By specifying  $RSRC\_B = 1$ , the rows of global matrix  $B$  are distributed over the process grid starting in the second row of the process grid. In addition, by specifying  $CSRC\_B = 1$ , the columns of global matrix  $B$  are distributed over the process grid starting in the second column of the process grid.

### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      N  NRHS  A  IA  JA  DESC_A  IPVT  B  IB  JB  DESC_B  INFO
      |  |    |  |  |  |      |  |  |  |  |      |
CALL PZGESV (9 , 5 , A , 1 , 1 , DESC_A , IPVT , B , 1 , 2 , DESC_B , INFO)
```

	Desc_A	Desc_B
DTYPE_	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	9	9

	Desc_A	Desc_B
N_	9	6
MB_	3	3
NB_	3	2
RSRC_	1	1
CSRC_	0	1
LLD_	See below <sup>2</sup>	See below <sup>2</sup>

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:  

$$\text{LLD\_A} = \text{MAX}(1, \text{NUMROC}(\text{M\_A}, \text{MB\_A}, \text{MYROW}, \text{RSRC\_A}, \text{NPROW}))$$

$$\text{LLD\_B} = \text{MAX}(1, \text{NUMROC}(\text{M\_B}, \text{MB\_B}, \text{MYROW}, \text{RSRC\_B}, \text{NPROW}))$$

In this example,  $\text{LLD\_A} = \text{LLD\_B} = 3$  on  $P_{00}$  and  $P_{01}$ , and  
 $\text{LLD\_A} = \text{LLD\_B} = 6$  on  $P_{10}$  and  $P_{11}$ .

Global general  $9 \times 9$  matrix  $A$  with block size  $3 \times 3$ :

B,D	0	1	2
0	$\begin{pmatrix} (2.0, 1.0) & (2.4, -1.0) & (2.8, -1.0) \\ (2.4, 1.0) & (2.0, 1.0) & (2.4, -1.0) \\ (2.8, 1.0) & (2.4, 1.0) & (2.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (3.2, -1.0) & (3.6, -1.0) & (4.0, -1.0) \\ (2.8, -1.0) & (3.2, -1.0) & (3.6, -1.0) \\ (2.4, -1.0) & (2.8, -1.0) & (3.2, -1.0) \end{pmatrix}$	$\begin{pmatrix} (4.4, -1.0) & (4.8, -1.0) & (5.2, -1.0) \\ (4.0, -1.0) & (4.4, -1.0) & (4.8, -1.0) \\ (3.6, -1.0) & (4.0, -1.0) & (4.4, -1.0) \end{pmatrix}$
1	$\begin{pmatrix} (3.2, 1.0) & (2.8, 1.0) & (2.4, 1.0) \\ (3.6, 1.0) & (3.2, 1.0) & (2.8, 1.0) \\ (4.0, 1.0) & (3.6, 1.0) & (3.2, 1.0) \end{pmatrix}$	$\begin{pmatrix} (2.0, 1.0) & (2.4, -1.0) & (2.8, -1.0) \\ (2.4, 1.0) & (2.0, 1.0) & (2.4, -1.0) \\ (2.8, 1.0) & (2.4, 1.0) & (2.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (3.2, -1.0) & (3.6, -1.0) & (4.0, -1.0) \\ (2.8, -1.0) & (3.2, -1.0) & (3.6, -1.0) \\ (2.4, -1.0) & (2.8, -1.0) & (3.2, -1.0) \end{pmatrix}$
2	$\begin{pmatrix} (4.4, 1.0) & (4.0, 1.0) & (3.6, 1.0) \\ (4.8, 1.0) & (4.4, 1.0) & (4.0, 1.0) \\ (5.2, 1.0) & (4.8, 1.0) & (4.4, 1.0) \end{pmatrix}$	$\begin{pmatrix} (3.2, 1.0) & (2.8, 1.0) & (2.4, 1.0) \\ (3.6, 1.0) & (3.2, 1.0) & (2.8, 1.0) \\ (4.0, 1.0) & (3.6, 1.0) & (3.2, 1.0) \end{pmatrix}$	$\begin{pmatrix} (2.0, 1.0) & (2.4, -1.0) & (2.8, -1.0) \\ (2.4, 1.0) & (2.0, 1.0) & (2.4, -1.0) \\ (2.8, 1.0) & (2.4, 1.0) & (2.0, 1.0) \end{pmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
1	$P_{00}$	$P_{01}$
0	$P_{10}$	$P_{11}$
2		

**Note:** The first row of  $A$  begins in the second row of the process grid.

Local arrays for  $A$ :

p,q	0	1
0	$\begin{pmatrix} (3.2, 1.0) & (2.8, 1.0) & (2.4, 1.0) \\ (3.6, 1.0) & (3.2, 1.0) & (2.8, 1.0) \\ (4.0, 1.0) & (3.6, 1.0) & (3.2, 1.0) \end{pmatrix}$	$\begin{pmatrix} (2.0, 1.0) & (2.4, -1.0) & (2.8, -1.0) \\ (2.4, 1.0) & (2.0, 1.0) & (2.4, -1.0) \\ (2.8, 1.0) & (2.4, 1.0) & (2.0, 1.0) \end{pmatrix}$
1	$\begin{pmatrix} (2.0, 1.0) & (2.4, -1.0) & (2.8, -1.0) \\ (2.4, 1.0) & (2.0, 1.0) & (2.4, -1.0) \\ (2.8, 1.0) & (2.4, 1.0) & (2.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (3.2, -1.0) & (3.6, -1.0) & (4.0, -1.0) \\ (2.8, -1.0) & (3.2, -1.0) & (3.6, -1.0) \\ (2.4, -1.0) & (2.8, -1.0) & (3.2, -1.0) \end{pmatrix}$

After the global matrix  $B$  is distributed over the process grid, only a portion of the global data structure is used—that is, global submatrix  $B$ . Following is the global



$9 \times 5$  submatrix  $B$ , starting at row 1 and column 2 in global general  $9 \times 6$  matrix  $B$  with block size  $3 \times 2$ :

B,D	0	1	2
0	$\begin{bmatrix} \cdot & (193.0, -10.6) \\ \cdot & (173.8, -9.4) \\ \cdot & (156.2, -5.4) \end{bmatrix}$	$\begin{bmatrix} (200.0, 21.8) & (207.0, 54.2) \\ (178.8, 20.2) & (183.8, 49.8) \\ (159.2, 22.2) & (162.2, 49.8) \end{bmatrix}$	$\begin{bmatrix} (214.0, 86.6) & (221.0, 119.0) \\ (188.8, 79.4) & (193.8, 109.0) \\ (165.2, 77.4) & (168.2, 105.0) \end{bmatrix}$
1	$\begin{bmatrix} \cdot & (141.0, 1.4) \\ \cdot & (129.0, 11.0) \\ \cdot & (121.0, 23.4) \end{bmatrix}$	$\begin{bmatrix} (142.0, 27.8) & (143.0, 54.2) \\ (128.0, 37.0) & (127.0, 63.0) \\ (118.0, 49.8) & (115.0, 76.2) \end{bmatrix}$	$\begin{bmatrix} (144.0, 80.6) & (145.0, 107.0) \\ (126.0, 89.0) & (125.0, 115.0) \\ (112.0, 102.6) & (109.0, 129.0) \end{bmatrix}$
2	$\begin{bmatrix} \cdot & (117.8, 38.6) \\ \cdot & (120.2, 56.6) \\ \cdot & (129.0, 77.4) \end{bmatrix}$	$\begin{bmatrix} (112.8, 66.2) & (107.8, 93.8) \\ (113.2, 86.2) & (106.2, 115.8) \\ (120.0, 109.8) & (111.0, 142.2) \end{bmatrix}$	$\begin{bmatrix} (102.8, 121.4) & (97.8, 149.0) \\ (99.2, 145.4) & (92.2, 175.0) \\ (102.0, 174.6) & (93.0, 207.0) \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	1	0 2
1	$P_{00}$	$P_{01}$
0 2	$P_{10}$	$P_{11}$

**Note:** The first row of  $B$  begins in the second row of the process grid, and the first column of  $B$  begins in the second column of the process grid.

Local arrays for  $B$ :

p,q	0	1
0	$\begin{bmatrix} (142.0, 27.8) & (143.0, 54.2) \\ (128.0, 37.0) & (127.0, 63.0) \\ (118.0, 49.8) & (115.0, 76.2) \end{bmatrix}$	$\begin{bmatrix} \cdot & (141.0, 1.4) & (144.0, 80.6) & (145.0, 107.0) \\ \cdot & (129.0, 11.0) & (126.0, 89.0) & (125.0, 115.0) \\ \cdot & (121.0, 23.4) & (112.0, 102.6) & (109.0, 129.0) \end{bmatrix}$
1	$\begin{bmatrix} (200.0, 21.8) & (207.0, 54.2) \\ (178.8, 20.2) & (183.8, 49.8) \\ (159.2, 22.2) & (162.2, 49.8) \\ (112.8, 66.2) & (107.8, 93.8) \\ (113.2, 86.2) & (106.2, 115.8) \\ (120.0, 109.8) & (111.0, 142.2) \end{bmatrix}$	$\begin{bmatrix} \cdot & (193.0, -10.6) & (214.0, 86.6) & (221.0, 119.0) \\ \cdot & (173.8, -9.4) & (188.8, 79.4) & (193.8, 109.0) \\ \cdot & (156.2, -5.4) & (165.2, 77.4) & (168.2, 105.0) \\ \cdot & (117.8, 38.6) & (102.8, 121.4) & (97.8, 149.0) \\ \cdot & (120.2, 56.6) & (99.2, 145.4) & (92.2, 175.0) \\ \cdot & (129.0, 77.4) & (102.0, 174.6) & (93.0, 207.0) \end{bmatrix}$

**Output:**

Global general  $9 \times 9$  transformed matrix  $A$  with block size  $3 \times 3$ :

B,D	0	1	2
0	$\begin{bmatrix} (5.2, 1.0) & (4.8, 1.0) & (4.4, 1.0) \\ (0.4, 0.1) & (0.6, -2.0) & (1.1, -1.9) \\ (0.5, 0.1) & (0.0, -0.1) & (0.6, -1.9) \end{bmatrix}$	$\begin{bmatrix} (4.0, 1.0) & (3.6, 1.0) & (3.2, 1.0) \\ (1.7, -1.9) & (2.3, -1.8) & (2.8, -1.8) \\ (1.2, -1.8) & (1.8, -1.7) & (2.5, -1.6) \end{bmatrix}$	$\begin{bmatrix} (2.8, 1.0) & (2.4, 1.0) & (2.0, 1.0) \\ (3.4, -1.7) & (3.9, -1.7) & (4.5, -1.6) \\ (3.1, -1.5) & (3.7, -1.4) & (4.3, -1.3) \end{bmatrix}$
1	$\begin{bmatrix} (0.6, 0.1) & (0.0, -0.1) & (-0.1, -0.1) \\ (0.6, 0.1) & (0.0, -0.1) & (-0.1, -0.1) \\ (0.7, 0.1) & (0.0, -0.1) & (0.0, 0.0) \end{bmatrix}$	$\begin{bmatrix} (0.7, -1.9) & (1.3, -1.7) & (2.0, -1.6) \\ (-0.1, 0.0) & (0.7, -1.9) & (1.5, -1.7) \\ (-0.1, 0.0) & (-0.1, 0.0) & (0.8, -1.9) \end{bmatrix}$	$\begin{bmatrix} (2.7, -1.5) & (3.4, -1.4) & (4.0, -1.2) \\ (2.2, -1.6) & (2.9, -1.5) & (3.7, -1.3) \\ (1.6, -1.8) & (2.4, -1.6) & (3.2, -1.5) \end{bmatrix}$
2	$\begin{bmatrix} (0.8, 0.0) & (0.0, 0.0) & (0.0, 0.0) \\ (0.9, 0.0) & (0.0, 0.0) & (0.0, 0.0) \\ (0.9, 0.0) & (0.0, 0.0) & (0.0, 0.0) \end{bmatrix}$	$\begin{bmatrix} (0.0, 0.0) & (0.0, 0.0) & (0.0, 0.0) \\ (0.0, 0.0) & (0.0, 0.0) & (0.0, 0.0) \\ (0.0, 0.0) & (0.0, 0.0) & (0.0, 0.0) \end{bmatrix}$	$\begin{bmatrix} (0.8, -1.9) & (1.7, -1.8) & (2.5, -1.8) \\ (0.0, 0.0) & (0.8, -2.0) & (1.7, -1.9) \\ (0.0, 0.0) & (0.0, 0.0) & (0.8, -2.0) \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

## PDGESV and PZGESV

B,D	0 2	1
1	$P_{00}$	$P_{01}$
0 2	$P_{10}$	$P_{11}$

**Note:** The first row of  $A$  begins in the second row of the process grid.

Local arrays for  $A$ :

p,q	0						1		
0	(0.6, 0.1)	(0.0,-0.1)	(-0.1,-0.1)	(2.7,-1.5)	(3.4,-1.4)	(4.0,-1.2)	(0.7,-1.9)	(1.3,-1.7)	(2.0,-1.6)
	(0.6, 0.1)	(0.0,-0.1)	(-0.1,-0.1)	(2.2,-1.6)	(2.9,-1.5)	(3.7,-1.3)	(-0.1, 0.0)	(0.7,-1.9)	(1.5,-1.7)
	(0.7, 0.1)	(0.0,-0.1)	(0.0, 0.0)	(1.6,-1.8)	(2.4,-1.6)	(3.2,-1.5)	(-0.1, 0.0)	(-0.1, 0.0)	(0.8,-1.9)
1	(5.2, 1.0)	(4.8, 1.0)	(4.4, 1.0)	(2.8, 1.0)	(2.4, 1.0)	(2.0, 1.0)	(4.0, 1.0)	(3.6, 1.0)	(3.2, 1.0)
	(0.4, 0.1)	(0.6,-2.0)	(1.1,-1.9)	(3.4,-1.7)	(3.9,-1.7)	(4.5,-1.6)	(1.7,-1.9)	(2.3,-1.8)	(2.8,-1.8)
	(0.5, 0.1)	(0.0,-0.1)	(0.6,-1.9)	(3.1,-1.5)	(3.7,-1.4)	(4.3,-1.3)	(1.2,-1.8)	(1.8,-1.7)	(2.5,-1.6)
	(0.8, 0.0)	(0.0, 0.0)	(0.0, 0.0)	(0.8,-1.9)	(1.7,-1.8)	(2.5,-1.8)	(0.0, 0.0)	(0.0, 0.0)	(0.0, 0.0)
	(0.9, 0.0)	(0.0, 0.0)	(0.0, 0.0)	(0.0, 0.0)	(0.8,-2.0)	(1.7,-1.9)	(0.0, 0.0)	(0.0, 0.0)	(0.0, 0.0)
	(0.9, 0.0)	(0.0, 0.0)	(0.0, 0.0)	(0.0, 0.0)	(0.0, 0.0)	(0.8,-2.0)	(0.0, 0.0)	(0.0, 0.0)	(0.0, 0.0)

Global vector *ipvt* of length 9 with block size 3:

B,D	0
0	9 9 9 -- 9 9 9 -- 9 9 9
1	9 9 9 -- 9 9 9 -- 9 9 9
2	9 9 9

**Note:** A copy of *ipvt* is distributed across each column of the process grid.

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
1	$P_{00}$	$P_{01}$
0 2	$P_{10}$	$P_{11}$

**Note:** The first row of *ipvt* begins in the second row of the process grid.

Local arrays for *ipvt*:

p,q	0	1
0	9 9 9	9 9 9
1	9 9 9 9 9 9	9 9 9 9 9 9

After the global matrix  $B$  is distributed over the process grid, only a portion of the global data structure is used—that is, global submatrix  $B$ . Following is the global  $9 \times 5$  submatrix  $B$ , starting at row 1 and column 2 in global general  $9 \times 6$  matrix  $B$  with block size  $3 \times 2$ :

B,D	0	1	2
0	<div> <div>.</div> <div>(1.0, 1.0)</div> </div> <div> <div>.</div> <div>(2.0, 1.0)</div> </div> <div> <div>.</div> <div>(3.0, 1.0)</div> </div>	<div> <div>(1.0, 2.0)</div> <div>(1.0, 3.0)</div> </div> <div> <div>(2.0, 2.0)</div> <div>(2.0, 3.0)</div> </div> <div> <div>(3.0, 2.0)</div> <div>(3.0, 3.0)</div> </div>	<div> <div>(1.0, 4.0)</div> <div>(1.0, 5.0)</div> </div> <div> <div>(2.0, 4.0)</div> <div>(2.0, 5.0)</div> </div> <div> <div>(3.0, 4.0)</div> <div>(3.0, 5.0)</div> </div>
1	<div> <div>.</div> <div>(4.0, 1.0)</div> </div> <div> <div>.</div> <div>(5.0, 1.0)</div> </div> <div> <div>.</div> <div>(6.0, 1.0)</div> </div>	<div> <div>(4.0, 2.0)</div> <div>(4.0, 3.0)</div> </div> <div> <div>(5.0, 2.0)</div> <div>(5.0, 3.0)</div> </div> <div> <div>(6.0, 2.0)</div> <div>(6.0, 3.0)</div> </div>	<div> <div>(4.0, 4.0)</div> <div>(4.0, 5.0)</div> </div> <div> <div>(5.0, 4.0)</div> <div>(5.0, 5.0)</div> </div> <div> <div>(6.0, 4.0)</div> <div>(6.0, 5.0)</div> </div>
2	<div> <div>.</div> <div>(7.0, 1.0)</div> </div> <div> <div>.</div> <div>(8.0, 1.0)</div> </div> <div> <div>.</div> <div>(9.0, 1.0)</div> </div>	<div> <div>(7.0, 2.0)</div> <div>(7.0, 3.0)</div> </div> <div> <div>(8.0, 2.0)</div> <div>(8.0, 3.0)</div> </div> <div> <div>(9.0, 2.0)</div> <div>(9.0, 3.0)</div> </div>	<div> <div>(7.0, 4.0)</div> <div>(7.0, 5.0)</div> </div> <div> <div>(8.0, 4.0)</div> <div>(8.0, 5.0)</div> </div> <div> <div>(9.0, 4.0)</div> <div>(9.0, 5.0)</div> </div>

The following is the  $2 \times 2$  process grid:

B,D	1	0 2
1	$P_{00}$	$P_{01}$
0 2	$P_{10}$	$P_{11}$

**Note:** The first row of  $B$  begins in the second row of the process grid, and the first column of  $B$  begins in the second column of the process grid.

Local arrays for  $B$ :

p,q	0	1
0	<div> <div>(3.0, 2.0)</div> <div>(3.0, 3.0)</div> </div> <div> <div>(4.0, 2.0)</div> <div>(4.0, 3.0)</div> </div> <div> <div>(5.0, 2.0)</div> <div>(5.0, 3.0)</div> </div>	<div> <div>.</div> <div>(3.0, 1.0)</div> <div>(3.0, 4.0)</div> <div>(3.0, 5.0)</div> </div> <div> <div>.</div> <div>(4.0, 1.0)</div> <div>(4.0, 4.0)</div> <div>(4.0, 5.0)</div> </div> <div> <div>.</div> <div>(5.0, 1.0)</div> <div>(5.0, 4.0)</div> <div>(5.0, 5.0)</div> </div>
1	<div> <div>(1.0, 2.0)</div> <div>(1.0, 3.0)</div> </div> <div> <div>(2.0, 2.0)</div> <div>(2.0, 3.0)</div> </div> <div> <div>(3.0, 2.0)</div> <div>(3.0, 3.0)</div> </div> <div> <div>(7.0, 2.0)</div> <div>(7.0, 3.0)</div> </div> <div> <div>(8.0, 2.0)</div> <div>(8.0, 3.0)</div> </div> <div> <div>(9.0, 2.0)</div> <div>(9.0, 3.0)</div> </div>	<div> <div>.</div> <div>(1.0, 1.0)</div> <div>(1.0, 4.0)</div> <div>(1.0, 5.0)</div> </div> <div> <div>.</div> <div>(2.0, 1.0)</div> <div>(2.0, 4.0)</div> <div>(2.0, 5.0)</div> </div> <div> <div>.</div> <div>(3.0, 1.0)</div> <div>(3.0, 4.0)</div> <div>(3.0, 5.0)</div> </div> <div> <div>.</div> <div>(7.0, 1.0)</div> <div>(7.0, 4.0)</div> <div>(7.0, 5.0)</div> </div> <div> <div>.</div> <div>(8.0, 1.0)</div> <div>(8.0, 4.0)</div> <div>(8.0, 5.0)</div> </div> <div> <div>.</div> <div>(9.0, 1.0)</div> <div>(9.0, 4.0)</div> <div>(9.0, 5.0)</div> </div>

The value of *info* is 0 on all processes.

## PDGETRF and PZGETRF — General Matrix Factorization

### Purpose

These subroutines factor general matrix  $A$  using Gaussian elimination with partial pivoting, *ipvt*, to compute the  $LU$  factorization of  $A$ , where, in this description:

- $A$  represents the global general submatrix  $A_{ia:ia+m-1, ja:ja+n-1}$  to be factored.
- *ipvt* represents the global vector *ipvt* <sub>$ia:ia+m-1$</sub>  containing the pivoting indices.
- $L$  is a lower triangular matrix.
- $U$  is an upper triangular matrix.

On output, the transformed matrix  $A$  contains  $U$  in the upper triangle (if  $m \geq n$ ) or upper trapezoid (if  $m < n$ ) and  $L$  in the strict lower triangle (if  $m \leq n$ ) or lower trapezoid (if  $m > n$ ). *ipvt* contains the pivots representing permutation  $P$ , such that  $A = PLU$ .

To solve the system of equations with any number of right-hand sides, follow the call to these subroutines with one or more calls to PDGETRS or PZGETRS, respectively.

If  $m = 0$  or  $n = 0$ , no computation is performed and the subroutine returns after doing some parameter checking. See references [17], [19], [23], [38], and [39].

Table 71. Data Types

$A$	<i>ipvt</i>	Subroutine
Long-precision real	Integer	PDGETRF
Long-precision complex	Integer	PZGETRF

### Syntax

Fortran	CALL PDGETRF   PZGETRF ( <i>m</i> , <i>n</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>ipvt</i> , <i>info</i> )
C and C++	pdgetrf   pzgetrf ( <i>m</i> , <i>n</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>ipvt</i> , <i>info</i> );

### On Entry

$m$  is the number of rows in submatrix  $A$  and the number of elements in vector *ipvt* used in the computation.

Scope: **global**

Specified as: a fullword integer;  $m \geq 0$ .

$n$  is the number of columns in submatrix  $A$  used in the computation.

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

$a$  is the local part of the global general matrix  $A$ , used in the system of equations. This identifies the **first element** of the local array  $A$ . This subroutine computes the location of the first element of the local subarray used, based on *ia*, *ja*, *desc\_a*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading LOCp( $ia+m-1$ ) by LOCq( $ja+n-1$ ) part of the local array  $A$  must contain the local pieces of the leading  $ia+m-1$  by  $ja+n-1$  part of the global matrix.

Scope: **local**

Specified as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 71 on page 370. Details about the square block-cyclic data distribution of global matrix *A* are stored in *desc\_a*.

*ia* is the row index of the global matrix *A*, identifying the first row of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq M\_A$  and  $ia+m-1 \leq M\_A$ .

*ja* is the column index of the global matrix *A*, identifying the first column of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$  and  $ja+n-1 \leq N\_A$ .

*desc\_a* is the array descriptor for global matrix *A*, described in the following table:

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $m = 0$ or $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	If $m = 0$ or $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_A < p$	Global
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < q$	Global
9	LLD_A	The leading dimension of the local array	$LLD\_A \geq \max(1, LOCp(M\_A))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

*ipvt* See On Return.

*info* See On Return.

### On Return

*a* is the updated local part of the global matrix *A*, containing the results of the factorization.

Scope: **local**

Returned as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 71 on page 370.

*ipvt* is the local part of the global vector *ipvt*, containing the pivot indices. This identifies the **first element** of the local array IPVT. This subroutine computes the location of the first element of the local subarray used, based on *ia*, *desc\_a*, *p*, and *myrow*; therefore, the leading  $\text{LOCp}(ia+m-1)$  part of the local array IPVT must contain the local pieces of the leading  $ia+m-1$  part of the global vector.

A copy of the vector *ipvt*, with a block size of MB\_A and global index *ia*, is returned to each column of the process grid. The process row over which the first row of *ipvt* is distributed is RSRC\_A.

Scope: **local**

Returned as: an array of (at least) length  $\text{LOCp}(ia+m-1)$ , containing fullword integers, where  $ia \leq (\text{pivoting indices}) \leq ia+m-1$ . Details about the block-cyclic data distribution of global vector *ipvt* are stored in *desc\_a*.

*info* has the following meaning:

If *info* = 0, global submatrix *A* is not singular, and the factorization completed normally.

If *info* > 0, global submatrix *A* is singular; that is, one or more columns of *L* and the corresponding diagonal of *U* contain all zeros. All columns of *L* are checked. *info* is set equal to *i*, the first column of *L* with a corresponding zero diagonal element, encountered at  $A_{ia+i-1, ja+i-1}$ . The factorization is completed; however, if you call PDGETRS/PZGETRS or PDGETRI/PZGETRI with these factors, results are unpredictable.

Scope: **global**

Returned as: a fullword integer; *info*  $\geq 0$ .

## Notes and Coding Rules

1. In your C program, argument *info* must be passed by reference.
2. The matrix and vector must have no common elements; otherwise, results are unpredictable.
3. The scalar data specified for input argument *n* must be the same for both PDGETRF/PZGETRF and PDGETRS/PZGETRS. In addition, the scalar data specified for input argument *m* in PDGETRF/PZGETRF **must be the same** as input argument *n* in both PDGETRF/PZGETRF and PDGETRS/PZGETRS.  
If, however, you do **not** plan to call PDGETRS/PZGETRS after calling PDGETRF/PZGETRF, then input arguments *m* and *n* in PDGETRF/PZGETRF do not need to be equal.
4. The global submatrices for *A* and *ipvt* input to PDGETRS/PZGETRS must be the same as for the corresponding output arguments for PDGETRF/PZGETRF; and thus, the scalar data specified for *ia*, *ja*, and the contents of *desc\_a* must also be the same.
5. The NUMROC utility subroutine can be used to determine the values of  $\text{LOCp}(M\_)$  and  $\text{LOCq}(N\_)$  used in the argument descriptions above. For details, see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28 and “NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process” on page 868.
6. The way these subroutines handle singularity differs from ScaLAPACK. These subroutines use the *info* argument to provide information about the singularity of *A*, like ScaLAPACK, but also provide an error message.
7. On both input and output, matrix *A* conforms to ScaLAPACK format.

8. The global general matrix  $A$  must be distributed using a square block-cyclic distribution; that is,  $MB\_A = NB\_A$ .
9. The global general matrix  $A$  must be aligned on a block row boundary; that is,  $ia-1$  must be a multiple of  $MB\_A$ .
10. The block row offset of  $A$  must be equal to the block column offset of  $A$ ; that is,  $\text{mod}(ia-1, MB\_A) = \text{mod}(ja-1, NB\_A)$ .
11. There is no array descriptor for *ipvt*. It is a column-distributed vector with block size  $MB\_A$ , local arrays of dimension  $LOCp(ia+m-1)$  by 1, and global index  $ia$ . A copy of this vector exists on each column of the process grid, and the process row over which the first column of *ipvt* is distributed is  $RSRC\_A$ .
12. For the Parallel ESSL SMP libraries, these subroutines use nonblocking collective communications; therefore, the `MP_SINGLE_THREAD` environment variable must be set to `NO` (which is the default value). See the *IBM Parallel Environment for AIX: MPI Programming Guide* for more information.

## Performance Considerations

1. For suggested block sizes, see “Coding Tips for Optimizing Parallel Performance” on page 77.
2. Pivoting imposes additional communication requirements over the process grid columns; therefore, you achieve optimal performance by using a process grid with  $p < q$ . On the other hand, a  $p \times 1$  grid provides the worse possible configuration.
3. For optimal performance, take the following items into consideration when choosing the  $NB\_A$  ( $= MB\_A$ ) value:
  - The cache size of the computational nodes.  $NB\_A$  determines the granularity of the most expensive part of the computation, which tends to increase the optimal value of  $NB\_A$ .
  - The communication and synchronization overhead. This has two aspects, the cost of internal synchronization points and the cost of broadcasts. These tend to slightly decrease the optimal value of  $NB\_A$ .
  - The model of communication adapter you are using.
  - Load balancing. For the best processor utilization, it is necessary for the processor nodes to be active for as long as possible; therefore, each one should have as many blocks as possible. For a given problem size, this tends to decrease the optimal value of  $NB\_A$  (best load balancing: 1) and is most relevant at very small problem sizes.
  - If  $NB\_A$  is equal to a power of 2, performance may be degraded.
  - Use the following rules of thumb for reasonably-sized problems:
    - For the SERIAL processors, choose  $NB\_A$  in the following range:
      - For PDGETRF, use [30, 50], avoiding 32.
      - For PZGETRF, use [10, 25], avoiding 16.
    - For the SMP processors, choose  $NB\_A$  in the following range:
      - For PDGETRF, use [70, 100].
      - For PZGETRF, use [30, 50], avoiding 32.

## Error Conditions

### Computational Errors

Matrix  $A$  is a singular matrix. For details, see the description of the *info* argument.

### Resource Errors

Unable to allocate work space

## Input-Argument and Miscellaneous Errors

### Stage 1:

1. DTYPE\_A is invalid.

### Stage 2:

1. CTEXT\_A is invalid.

### Stage 3:

1. This subroutine was called from outside the process grid.

### Stage 4:

1.  $m < 0$
2.  $n < 0$
3.  $M\_A < 0$  and  $(m = 0 \text{ or } n = 0)$ ;  $M\_A < 1$  otherwise
4.  $N\_A < 0$  and  $(m = 0 \text{ or } n = 0)$ ;  $N\_A < 1$  otherwise
5.  $ia < 1$
6.  $ja < 1$
7.  $MB\_A < 1$
8.  $NB\_A < 1$
9.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
10.  $CSRC\_A < 0$  or  $CSRC\_A \geq q$

### Stage 5: If $m \neq 0$ and $n \neq 0$ :

1.  $ia > M\_A$
  2.  $ja > N\_A$
  3.  $ia+m-1 > M\_A$
  4.  $ja+n-1 > N\_A$
- In all cases:
5.  $MB\_A \neq NB\_A$
  6.  $\text{mod}(ia-1, MB\_A) \neq \text{mod}(ja-1, NB\_A)$
  7.  $\text{mod}(ia-1, MB\_A) \neq 0$

### Stage 6:

1.  $LLD\_A < \max(1, \text{LOCp}(M\_A))$   
Each of the following global input arguments are checked to determine whether its value differs from the value specified on process  $P_{00}$ :
2.  $m$  differs.
3.  $n$  differs.
4.  $ia$  differs.
5.  $ja$  differs.
6. DTYPE\_A differs.
7. M\_A differs.
8. N\_A differs.
9. MB\_A differs.
10. NB\_A differs.
11. RSRC\_A differs.
12. CSRC\_A differs.

## Examples

### Example 1

This example factors a  $9 \times 9$  real general matrix using a  $2 \times 2$  process grid. By specifying  $RSRC\_A = 1$ , the rows of global matrix  $A$  and the elements of global vector  $ipvt$  are distributed over the process grid starting in the second row of the process grid.



**Call Statements and Input:**

```

ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      M      N      A      IA      JA      DESC_A      IPVT      INFO
      |      |      |      |      |      |      |      |
CALL PDGETRF( 9 , 9 , A , 1 , 1 , DESC_A , IPVT , INFO )

```

	Desc_A
DTYPE_	1
CTXT_	<i>icontxt</i> <sup>1</sup>
M_	9
N_	9
MB_	3
NB_	3
RSRC_	1
CSRC_	0
LLD_	See below <sup>2</sup>

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:  

$$\text{LLD\_A} = \text{MAX}(1, \text{NUMROC}(\text{M\_A}, \text{MB\_A}, \text{MYROW}, \text{RSRC\_A}, \text{NPROW}))$$
In this example,  $\text{LLD\_A} = 3$  on  $P_{00}$  and  $P_{01}$ , and  $\text{LLD\_A} = 6$  on  $P_{10}$  and  $P_{11}$ .

Global general  $9 \times 9$  matrix  $A$  with block size  $3 \times 3$ :

B,D	0	1	2
0	$\begin{bmatrix} 1.0 & 1.2 & 1.4 \\ 1.2 & 1.0 & 1.2 \\ 1.4 & 1.2 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 1.6 & 1.8 & 2.0 \\ 1.4 & 1.6 & 1.8 \\ 1.2 & 1.4 & 1.6 \end{bmatrix}$	$\begin{bmatrix} 2.2 & 2.4 & 2.6 \\ 2.0 & 2.2 & 2.4 \\ 1.8 & 2.0 & 2.2 \end{bmatrix}$
1	$\begin{bmatrix} 1.6 & 1.4 & 1.2 \\ 1.8 & 1.6 & 1.4 \\ 2.0 & 1.8 & 1.6 \end{bmatrix}$	$\begin{bmatrix} 1.0 & 1.2 & 1.4 \\ 1.2 & 1.0 & 1.2 \\ 1.4 & 1.2 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 1.6 & 1.8 & 2.0 \\ 1.4 & 1.6 & 1.8 \\ 1.2 & 1.4 & 1.6 \end{bmatrix}$
2	$\begin{bmatrix} 2.2 & 2.0 & 1.8 \\ 2.4 & 2.2 & 2.0 \\ 2.6 & 2.4 & 2.2 \end{bmatrix}$	$\begin{bmatrix} 1.6 & 1.4 & 1.2 \\ 1.8 & 1.6 & 1.4 \\ 2.0 & 1.8 & 1.6 \end{bmatrix}$	$\begin{bmatrix} 1.0 & 1.2 & 1.4 \\ 1.2 & 1.0 & 1.2 \\ 1.4 & 1.2 & 1.0 \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
1	$P_{00}$	$P_{01}$
0	$P_{10}$	$P_{11}$
2		

**Note:** The first row of  $A$  begins in the second row of the process grid.

Local arrays for  $A$ :

## PDGETRF and PZGETRF

p,q	0						1		
0	1.6	1.4	1.2	1.6	1.8	2.0	1.0	1.2	1.4
	1.8	1.6	1.4	1.4	1.6	1.8	1.2	1.0	1.2
	2.0	1.8	1.6	1.2	1.4	1.6	1.4	1.2	1.0
1	1.0	1.2	1.4	2.2	2.4	2.6	1.6	1.8	2.0
	1.2	1.0	1.2	2.0	2.2	2.4	1.4	1.6	1.8
	1.4	1.2	1.0	1.8	2.0	2.2	1.2	1.4	1.6
	2.2	2.0	1.8	1.0	1.2	1.4	1.6	1.4	1.2
	2.4	2.2	2.0	1.2	1.0	1.2	1.8	1.6	1.4
	2.6	2.4	2.2	1.4	1.2	1.0	2.0	1.8	1.6

**Output:**

Global general  $9 \times 9$  transformed matrix  $A$  with block size  $3 \times 3$ :

B,D	0			1			2		
0	2.6	2.4	2.2	2.0	1.8	1.6	1.4	1.2	1.0
	0.4	0.3	0.6	0.8	1.1	1.4	1.7	1.9	2.2
	0.5	-0.4	0.4	0.8	1.2	1.6	2.0	2.4	2.8
1	0.5	-0.3	0.0	0.4	0.8	1.2	1.6	2.0	2.4
	0.6	-0.3	0.0	0.0	0.4	0.8	1.2	1.6	2.0
	0.7	-0.2	0.0	0.0	0.0	0.4	0.8	1.2	1.6
2	0.8	-0.2	0.0	0.0	0.0	0.0	0.4	0.8	1.2
	0.8	-0.1	0.0	0.0	0.0	0.0	0.0	0.4	0.8
	0.9	-0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.4

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
1	$P_{00}$	$P_{01}$
0	$P_{10}$	$P_{11}$
2		

**Note:** The first row of  $A$  begins in the second row of the process grid.

Local arrays for  $A$ :

p,q	0						1		
0	0.5	-0.3	0.0	1.6	2.0	2.4	0.4	0.8	1.2
	0.6	-0.3	0.0	1.2	1.6	2.0	0.0	0.4	0.8
	0.7	-0.2	0.0	0.8	1.2	1.6	0.0	0.0	0.4
1	2.6	2.4	2.2	1.4	1.2	1.0	2.0	1.8	1.6
	0.4	0.3	0.6	1.7	1.9	2.2	0.8	1.1	1.4
	0.5	-0.4	0.4	2.0	2.4	2.8	0.8	1.2	1.6
	0.8	-0.2	0.0	0.4	0.8	1.2	0.0	0.0	0.0
	0.8	-0.1	0.0	0.0	0.4	0.8	0.0	0.0	0.0
	0.9	-0.1	0.0	0.0	0.0	0.4	0.0	0.0	0.0

Global vector *ipvt* of length 9 with block size 3:

B,D	0
0	9
	9
	9
	--
	9

1	9
	9
	--
	9
2	9
	9

**Note:** A copy of *ipvt* is distributed across each column of the process grid.

The following is the 2 × 2 process grid:

B,D		
----	-----	-----
1	P <sub>00</sub>	P <sub>01</sub>
----	-----	-----
0	P <sub>10</sub>	P <sub>11</sub>
2		

**Note:** The first row of *ipvt* begins in the second row of the process grid.

Local arrays for *ipvt*:

p,q	0	1
----	-----	-----
	9	9
	9	9
0	9	9
----	-----	-----
	9	9
	9	9
	9	9
1	9	9
	9	9
	9	9

The value of *info* is 0 on all processes.

**Example 2**

This example factors a 9 × 9 complex matrix using a 2 × 2 process grid. By specifying RSRC\_A = 1, the rows of global matrix *A* and the elements of global vector *ipvt* are distributed over the process grid starting in the second row of the process grid.

**Call Statements and Input:**

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      M      N      A      IA      JA      DESC_A      IPVT      INFO
      |      |      |      |      |      |      |      |
CALL PZGETRF( 9 , 9 , A , 1 , 1 ,  , DESC_A , IPVT , INFO )
```

	Desc_A
DTYPE_	1
CTXT_	<i>icontxt</i> <sup>1</sup>
M_	9
N_	9

## PDGETRF and PZGETRF

	Desc_A
MB_	3
NB_	3
RSRC_	1
CSRC_	0
LLD_	See below <sup>2</sup>

### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.

2. Each process should set the LLD\_ as follows:

$$\text{LLD\_A} = \text{MAX}(1, \text{NUMROC}(\text{M\_A}, \text{MB\_A}, \text{MYROW}, \text{RSRC\_A}, \text{NPROW}))$$

In this example, LLD\_A = 3 on P<sub>00</sub> and P<sub>01</sub>, and LLD\_A = 6 on P<sub>10</sub> and P<sub>11</sub>.

Global general  $9 \times 9$  matrix *A* with block size  $3 \times 3$ :

B,D	0	1	2
0	(2.0, 1.0) (2.4,-1.0) (2.8,-1.0) (2.4, 1.0) (2.0, 1.0) (2.4,-1.0) (2.8, 1.0) (2.4, 1.0) (2.0, 1.0)	(3.2,-1.0) (3.6,-1.0) (4.0,-1.0) (2.8,-1.0) (3.2,-1.0) (3.6,-1.0) (2.4,-1.0) (2.8,-1.0) (3.2,-1.0)	(4.4,-1.0) (4.8,-1.0) (5.2,-1.0) (4.0,-1.0) (4.4,-1.0) (4.8,-1.0) (3.6,-1.0) (4.0,-1.0) (4.4,-1.0)
1	(3.2, 1.0) (2.8, 1.0) (2.4, 1.0) (3.6, 1.0) (3.2, 1.0) (2.8, 1.0) (4.0, 1.0) (3.6, 1.0) (3.2, 1.0)	(2.0, 1.0) (2.4,-1.0) (2.8,-1.0) (2.4, 1.0) (2.0, 1.0) (2.4,-1.0) (2.8, 1.0) (2.4, 1.0) (2.0, 1.0)	(3.2,-1.0) (3.6,-1.0) (4.0,-1.0) (2.8,-1.0) (3.2,-1.0) (3.6,-1.0) (2.4,-1.0) (2.8,-1.0) (3.2,-1.0)
2	(4.4, 1.0) (4.0, 1.0) (3.6, 1.0) (4.8, 1.0) (4.4, 1.0) (4.0, 1.0) (5.2, 1.0) (4.8, 1.0) (4.4, 1.0)	(3.2, 1.0) (2.8, 1.0) (2.4, 1.0) (3.6, 1.0) (3.2, 1.0) (2.8, 1.0) (4.0, 1.0) (3.6, 1.0) (3.2, 1.0)	(2.0, 1.0) (2.4,-1.0) (2.8,-1.0) (2.4, 1.0) (2.0, 1.0) (2.4,-1.0) (2.8, 1.0) (2.4, 1.0) (2.0, 1.0)

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
1	P <sub>00</sub>	P <sub>01</sub>
0	P <sub>10</sub>	P <sub>11</sub>
2		

**Note:** The first row of *A* begins in the second row of the process grid.

Local arrays for *A*:

p,q	0	1
0	(3.2, 1.0) (2.8, 1.0) (2.4, 1.0) (3.2,-1.0) (3.6,-1.0) (4.0,-1.0) (3.6, 1.0) (3.2, 1.0) (2.8, 1.0) (2.8,-1.0) (3.2,-1.0) (3.6,-1.0) (4.0, 1.0) (3.6, 1.0) (3.2, 1.0) (2.4,-1.0) (2.8,-1.0) (3.2,-1.0)	(2.0, 1.0) (2.4,-1.0) (2.8,-1.0) (2.4, 1.0) (2.0, 1.0) (2.4,-1.0) (2.8, 1.0) (2.4, 1.0) (2.0, 1.0)
1	(2.0, 1.0) (2.4,-1.0) (2.8,-1.0) (4.4,-1.0) (4.8,-1.0) (5.2,-1.0) (2.4, 1.0) (2.0, 1.0) (2.4,-1.0) (4.0,-1.0) (4.4,-1.0) (4.8,-1.0) (2.8, 1.0) (2.4, 1.0) (2.0, 1.0) (3.6,-1.0) (4.0,-1.0) (4.4,-1.0) (4.4, 1.0) (4.0, 1.0) (3.6, 1.0) (2.0, 1.0) (2.4,-1.0) (2.8,-1.0) (4.8, 1.0) (4.4, 1.0) (4.0, 1.0) (2.4, 1.0) (2.0, 1.0) (2.4,-1.0) (5.2, 1.0) (4.8, 1.0) (4.4, 1.0) (2.8, 1.0) (2.4, 1.0) (2.0, 1.0)	(3.2,-1.0) (3.6,-1.0) (4.0,-1.0) (2.8,-1.0) (3.2,-1.0) (3.6,-1.0) (2.4,-1.0) (2.8,-1.0) (3.2,-1.0) (3.2, 1.0) (2.8, 1.0) (2.4, 1.0) (3.6, 1.0) (3.2, 1.0) (2.8, 1.0) (4.0, 1.0) (3.6, 1.0) (3.2, 1.0)

### Output:

Global general  $9 \times 9$  transformed matrix *A* with block size  $3 \times 3$ :

B,D	0			1			2		
0	(5.2, 1.0)	(4.8, 1.0)	(4.4, 1.0)	(4.0, 1.0)	(3.6, 1.0)	(3.2, 1.0)	(2.8, 1.0)	(2.4, 1.0)	(2.0, 1.0)
	(0.4, 0.1)	(0.6,-2.0)	(1.1,-1.9)	(1.7,-1.9)	(2.3,-1.8)	(2.8,-1.8)	(3.4,-1.7)	(3.9,-1.7)	(4.5,-1.6)
	(0.5, 0.1)	(0.0,-0.1)	(0.6,-1.9)	(1.2,-1.8)	(1.8,-1.7)	(2.5,-1.6)	(3.1,-1.5)	(3.7,-1.4)	(4.3,-1.3)
1	(0.6, 0.1)	(0.0,-0.1)	(-0.1,-0.1)	(0.7,-1.9)	(1.3,-1.7)	(2.0,-1.6)	(2.7,-1.5)	(3.4,-1.4)	(4.0,-1.2)
	(0.6, 0.1)	(0.0,-0.1)	(-0.1,-0.1)	(-0.1, 0.0)	(0.7,-1.9)	(1.5,-1.7)	(2.2,-1.6)	(2.9,-1.5)	(3.7,-1.3)
	(0.7, 0.1)	(0.0,-0.1)	(0.0, 0.0)	(-0.1, 0.0)	(-0.1, 0.0)	(0.8,-1.9)	(1.6,-1.8)	(2.4,-1.6)	(3.2,-1.5)
2	(0.8, 0.0)	(0.0, 0.0)	(0.0, 0.0)	(0.0, 0.0)	(0.0, 0.0)	(0.0, 0.0)	(0.8,-1.9)	(1.7,-1.8)	(2.5,-1.8)
	(0.9, 0.0)	(0.0, 0.0)	(0.0, 0.0)	(0.0, 0.0)	(0.0, 0.0)	(0.0, 0.0)	(0.0, 0.0)	(0.8,-2.0)	(1.7,-1.9)
	(0.9, 0.0)	(0.0, 0.0)	(0.0, 0.0)	(0.0, 0.0)	(0.0, 0.0)	(0.0, 0.0)	(0.0, 0.0)	(0.0, 0.0)	(0.8,-2.0)

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
1	P <sub>00</sub>	P <sub>01</sub>
0 2	P <sub>10</sub>	P <sub>11</sub>

**Note:** The first row of *A* begins in the second row of the process grid.

Local arrays for *A*:

p,q	0						1		
0	(0.6, 0.1)	(0.0,-0.1)	(-0.1,-0.1)	(2.7,-1.5)	(3.4,-1.4)	(4.0,-1.2)	(0.7,-1.9)	(1.3,-1.7)	(2.0,-1.6)
	(0.6, 0.1)	(0.0,-0.1)	(-0.1,-0.1)	(2.2,-1.6)	(2.9,-1.5)	(3.7,-1.3)	(-0.1, 0.0)	(0.7,-1.9)	(1.5,-1.7)
	(0.7, 0.1)	(0.0,-0.1)	(0.0, 0.0)	(1.6,-1.8)	(2.4,-1.6)	(3.2,-1.5)	(-0.1, 0.0)	(-0.1, 0.0)	(0.8,-1.9)
1	(5.2, 1.0)	(4.8, 1.0)	(4.4, 1.0)	(2.8, 1.0)	(2.4, 1.0)	(2.0, 1.0)	(4.0, 1.0)	(3.6, 1.0)	(3.2, 1.0)
	(0.4, 0.1)	(0.6,-2.0)	(1.1,-1.9)	(3.4,-1.7)	(3.9,-1.7)	(4.5,-1.6)	(1.7,-1.9)	(2.3,-1.8)	(2.8,-1.8)
	(0.5, 0.1)	(0.0,-0.1)	(0.6,-1.9)	(3.1,-1.5)	(3.7,-1.4)	(4.3,-1.3)	(1.2,-1.8)	(1.8,-1.7)	(2.5,-1.6)
	(0.8, 0.0)	(0.0, 0.0)	(0.0, 0.0)	(0.8,-1.9)	(1.7,-1.8)	(2.5,-1.8)	(0.0, 0.0)	(0.0, 0.0)	(0.0, 0.0)
	(0.9, 0.0)	(0.0, 0.0)	(0.0, 0.0)	(0.0, 0.0)	(0.8,-2.0)	(1.7,-1.9)	(0.0, 0.0)	(0.0, 0.0)	(0.0, 0.0)
	(0.9, 0.0)	(0.0, 0.0)	(0.0, 0.0)	(0.0, 0.0)	(0.0, 0.0)	(0.8,-2.0)	(0.0, 0.0)	(0.0, 0.0)	(0.0, 0.0)
	(0.9, 0.0)	(0.0, 0.0)	(0.0, 0.0)	(0.0, 0.0)	(0.0, 0.0)	(0.8,-2.0)	(0.0, 0.0)	(0.0, 0.0)	(0.0, 0.0)

Global vector *ipvt* of length 9 with block size 3:

B,D	0
0	9
	9
	9
1	--
	9
	9
2	--
	9
	9

**Note:** A copy of *ipvt* is distributed across each column of the process grid.

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
1	P <sub>00</sub>	P <sub>01</sub>
0 2	P <sub>10</sub>	P <sub>11</sub>

**Note:** The first row of *ipvt* begins in the second row of the process grid.

## PDGETRF and PZGETRF

Local arrays for *ipvt*:

p,q	0	1
0	9	9
	9	9
	9	9
1	9	9
	9	9
	9	9
	9	9
	9	9

The value of *info* is 0 on all processes.

## PDGETRS and PZGETRS — General Matrix Solve

### Purpose

PDGETRS solves one of the following systems of equations for multiple right-hand sides:

- 1.  $AX = B$
- 2.  $A^T X = B$

PZGETRS solves one of the following systems of equations for multiple right-hand sides:

- 1.  $AX = B$
- 2.  $A^T X = B$
- 3.  $A^H X = B$

In the formulas above:

- $A$  represents the global general submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  containing the  $LU$  factorization.
- $B$  represents the global general submatrix  $B_{ib:ib+n-1, jb:jb+nrhs-1}$  containing the right-hand sides in its columns.
- $X$  represents the global general submatrix  $B_{ib:ib+n-1, jb:jb+nrhs-1}$  containing the solution vectors in its columns.

These subroutines use the results of the factorization of matrix  $A$ , produced by a preceding call to PDGETRF or PZGETRF, respectively. For details on the factorization, see “PDGETRF and PZGETRF — General Matrix Factorization” on page 370.

If  $n = 0$  or  $nrhs = 0$ , no computation is performed and the subroutine returns after doing some parameter checking. See references [17], [19], [23], [38], and [39].

Table 72. Data Types

$A, B$	<i>ipvt</i>	Subroutine
Long-precision real	Integer	PDGETRS
Long-precision complex	Integer	PZGETRS

### Syntax

<b>Fortran</b>	CALL PDGETRS   PZGETRS ( <i>transa</i> , <i>n</i> , <i>nrhs</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>ipvt</i> , <i>b</i> , <i>ib</i> , <i>jb</i> , <i>desc_b</i> , <i>info</i> )
<b>C and C++</b>	pdgetrs   pzgetrs ( <i>transa</i> , <i>n</i> , <i>nrhs</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>ipvt</i> , <i>b</i> , <i>ib</i> , <i>jb</i> , <i>desc_b</i> , <i>info</i> );

### On Entry

*transa* indicates the form of matrix  $A$  to use in the computation, where:

If *transa* = 'N',  $A$  is used in the computation, resulting in solution 1.

If *transa* = 'T',  $A^T$  is used in the computation, resulting in solution 2.

If *transa* = 'C',  $A^H$  is used in the computation, resulting in solution 3.

Scope: **global**

Specified as: a single character; *transa* = 'N', 'T', or 'C'.

*n* is the order of the factored submatrix  $A$  and the number of rows in submatrix  $B$ .

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

*nrhs* is the number of right-hand sides— that is, the number of columns in submatrix *B* used in the computation.

Scope: **global**

Specified as: a fullword integer;  $nrhs \geq 0$ .

*a* is the local part of the global general matrix *A*, containing the factorization of matrix *A* produced by a preceding call to PDGETRF or PZGETRF, respectively. This identifies the **first element** of the local array *A*. This subroutine computes the location of the first element of the local subarray used, based on *ia*, *ja*, *desc\_a*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading LOCp(*ia*+*n*−1) by LOCq(*ja*+*n*−1) part of the local array *A* must contain the local pieces of the leading *ia*+*n*−1 by *ja*+*n*−1 part of the global matrix.

Scope: **local**

Specified as: an LLD\_ *A* by (at least) LOCq(*N\_**A*) array, containing numbers of the data type indicated in Table 72 on page 381. Details about the square block-cyclic data distribution of global matrix *A* are stored in *desc\_a*.

*ia* is the row index of the global matrix *A*, identifying the first row of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq M\_A$  and  $ia+n-1 \leq M\_A$ .

*ja* is the column index of the global matrix *A*, identifying the first column of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$  and  $ja+n-1 \leq N\_A$ .

*desc\_a* is the array descriptor for global matrix *A*, described in the following table:

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_ <i>A</i>	Descriptor type	DTYPE_ <i>A</i> =1	Global
2	CTXT_ <i>A</i>	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	<i>M_</i> <i>A</i>	Number of rows in the global matrix	If $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	<i>N_</i> <i>A</i>	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	<i>MB_</i> <i>A</i>	Row block size	$MB\_A \geq 1$	Global
6	<i>NB_</i> <i>A</i>	Column block size	$NB\_A \geq 1$	Global
7	<i>RSRC_</i> <i>A</i>	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_A < p$	Global



<i>desc_a</i>	Name	Description	Limits	Scope
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq \text{CSRC\_A} < q$	Global
9	LLD_A	The leading dimension of the local array	$\text{LLD\_A} \geq \max(1, \text{LOCp}(\text{M\_A}))$	Local

Specified as: an array of (at least) length 9, containing fullword integers.

*ipvt* is the local part of the global vector *ipvt*, containing the pivoting indices produced on a preceding call to PDGETRF or PZGETRF, respectively. This identifies the **first element** of the local array IPVT. This subroutine computes the location of the first element of the local subarray used, based on *ia*, *desc\_a*, *p*, and *myrow*; therefore, the leading  $\text{LOCp}(ia+n-1)$  part of the local array IPVT must contain the local pieces of the leading  $ia+n-1$  part of the global vector.

A copy of the vector *ipvt*, with a block size of MB\_A and global index *ia*, is contained in each column of the process grid. The process row over which the first row of *ipvt* is distributed is RSRC\_A.

Scope: **local**

Specified as: an array of (at least) length  $\text{LOCp}(ia+n-1)$ , containing fullword integers, where  $ia \leq (\text{pivoting index values}) \leq ia+n-1$ . Details about the block-cyclic data distribution of global vector *ipvt* are stored in *desc\_a*.

*b* is the local part of the global general matrix *B*, containing the right-hand sides of the system. This identifies the **first element** of the local array B. This subroutine computes the location of the first element of the local subarray used, based on *ib*, *jb*, *desc\_b*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading  $\text{LOCp}(ib+n-1)$  by  $\text{LOCq}(jb+nrhs-1)$  part of the local array B must contain the local pieces of the leading  $ib+n-1$  by  $jb+nrhs-1$  part of the global matrix.

Scope: **local**

Specified as: an LLD\_B by (at least)  $\text{LOCq}(\text{N\_B})$  array, containing numbers of the data type indicated in Table 72 on page 381. Details about the block-cyclic data distribution of global matrix *B* are stored in *desc\_b*.

*ib* is the row index of the global matrix *B*, identifying the first row of the submatrix *B*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ib \leq \text{M\_B}$  and  $ib+n-1 \leq \text{M\_B}$ .

*jb* is the column index of the global matrix *B*, identifying the first column of the submatrix *B*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq jb \leq \text{N\_B}$  and  $jb+nrhs-1 \leq \text{N\_B}$ .

*desc\_b* is the array descriptor for global matrix *B*, described in the following table:

<i>desc_b</i>	Name	Description	Limits	Scope
1	DTYPE_B	Descriptor type	$\text{DTYPE\_B}=1$	Global

## PDGETRS and PZGETRS

<i>desc_b</i>	Name	Description	Limits	Scope
2	CTXT_B	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_B	Number of rows in the global matrix	If $n = 0$ or $nrhs = 0$ : $M\_B \geq 0$ Otherwise: $M\_B \geq 1$	Global
4	N_B	Number of columns in the global matrix	If $n = 0$ or $nrhs = 0$ : $N\_B \geq 0$ Otherwise: $N\_B \geq 1$	Global
5	MB_B	Row block size	$MB\_B \geq 1$	Global
6	NB_B	Column block size	$NB\_B \geq 1$	Global
7	RSRC_B	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_B < p$	Global
8	CSRC_B	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_B < q$	Global
9	LLD_B	The leading dimension of the local array	$LLD\_B \geq \max(1, LOCp(M\_B))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

*info* See On Return.

### On Return

*b* is the updated local part of the global matrix *B*, containing the solution vectors.

Scope: **local**

Returned as: an LLD\_B by (at least) LOCq(N\_B) array, containing numbers of the data type indicated in Table 72 on page 381.

*info* indicates that a successful computation occurred.

Scope: **global**

Returned as: a fullword integer; *info* = 0.

## Notes and Coding Rules

1. In your C program, argument *info* must be passed by reference.
2. This subroutine accepts lowercase letters for the *transa* argument.
3. For PDGETRS, if you specify 'C' for the *transa* argument, it is interpreted as though you specified 'T'.
4. The matrices and vector must have no common elements; otherwise, results are unpredictable.
5. The scalar data specified for input argument *n* must be the same for both PDGETRF/PZGETRF and PDGETRS/PZGETRS. In addition, the scalar data specified for input argument *m* in PDGETRF/PZGETRF **must be the same** as input argument *n* in both PDGETRF/PZGETRF and PDGETRS/PZGETRS.

If, however, you do **not** plan to call PDGETRS/PZGETRS after calling PDGETRF/PZGETRF, then input arguments  $m$  and  $n$  in PDGETRF/PZGETRF do not need to be equal.

6. The global submatrices for  $A$  and *ipvt* input to PDGETRS/PZGETRS must be the same as for the corresponding output arguments for PDGETRF/PZGETRF; and thus, the scalar data specified for  $ia$ ,  $ja$ , and the contents of *desc\_a* must also be the same.
7. The NUMROC utility subroutine can be used to determine the values of LOCp(M\_) and LOCq(N\_) used in the argument descriptions above. For details, see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28 and “NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process” on page 868.
8. For suggested block sizes, see “Coding Tips for Optimizing Parallel Performance” on page 77.
9. On both input and output, matrices  $A$  and  $B$  conform to ScaLAPACK format.
10. The following values must be equal: CTXT\_A = CTXT\_B.
11. The global general matrix  $A$  must be distributed using a square block-cyclic distribution; that is, MB\_A = NB\_A.
12. The following block sizes must be equal: MB\_A = MB\_B.
13. The global general matrix  $A$  must be aligned on a block row boundary; that is,  $ia-1$  must be a multiple of MB\_A.
14. The block row offset of  $A$  must be equal to the block column offset of  $A$ ; that is,  $\text{mod}(ia-1, MB_A) = \text{mod}(ja-1, NB_A)$ .
15. The block row offset of  $A$  must be equal to the block row offset of  $B$ ; that is,  $\text{mod}(ia-1, MB_A) = \text{mod}(ib-1, MB_B)$ .
16. In the process grid, the process row containing the first row of the submatrix  $A$  must also contain the first row of the submatrix  $B$ ; that is,  $iarow = ibrow$ , where:
  - $iarow = \text{mod}((((ia-1)/MB_A)+RSRC_A), p)$
  - $ibrow = \text{mod}((((ib-1)/MB_B)+RSRC_B), p)$
17. There is no array descriptor for *ipvt*. It is a column-distributed vector with block size MB\_A, local arrays of dimension LOCp( $ia+n-1$ ) by 1, and global index  $ia$ . A copy of this vector exists on each column of the process grid, and the process row over which the first column of *ipvt* is distributed is RSRC\_A.

## Error Conditions

### Computational Errors

None

**Note:** If the factorization performed by PDGETRF/PZGETRF failed because of a singular matrix  $A$ , the results returned by this subroutine are unpredictable. For details, see the *info* output argument for PDGETRF/PZGETRF.

### Resource Errors

Unable to allocate work space

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. DTYPE\_A is invalid.
2. DTYPE\_B is invalid.

## Stage 2:

1. CTXT\_A is invalid.

## Stage 3:

1. This subroutine was called from outside the process grid.

## Stage 4:

1.  $transa \neq 'N', 'T', \text{ or } 'C'$
2.  $n < 0$
3.  $nrhs < 0$
4.  $M\_A < 0$  and  $n = 0$ ;  $M\_A < 1$  otherwise
5.  $N\_A < 0$  and  $n = 0$ ;  $N\_A < 1$  otherwise
6.  $ia < 1$
7.  $ja < 1$
8.  $MB\_A < 1$
9.  $NB\_A < 1$
10.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
11.  $CSRC\_A < 0$  or  $CSRC\_A \geq q$
12.  $M\_B < 0$  and  $(n = 0 \text{ or } nrhs = 0)$ ;  $M\_B < 1$  otherwise
13.  $N\_B < 0$  and  $(n = 0 \text{ or } nrhs = 0)$ ;  $N\_B < 1$  otherwise
14.  $ib < 1$
15.  $jb < 1$
16.  $MB\_B < 1$
17.  $NB\_B < 1$
18.  $RSRC\_B < 0$  or  $RSRC\_B \geq p$
19.  $CSRC\_B < 0$  or  $CSRC\_B \geq q$
20.  $CTXT\_A \neq CTXT\_B$

## Stage 5: If $n \neq 0$ :

1.  $ia > M\_A$
  2.  $ja > N\_A$
  3.  $ia+n-1 > M\_A$
  4.  $ja+n-1 > N\_A$
- If  $n \neq 0$  and  $nrhs \neq 0$ :
5.  $ib > M\_B$
  6.  $jb > N\_B$
  7.  $ib+n-1 > M\_B$
  8.  $jb+nrhs-1 > N\_B$
- In all cases:
9.  $MB\_A \neq NB\_A$
  10.  $\text{mod}(ia-1, MB\_A) \neq \text{mod}(ja-1, NB\_A)$
  11.  $MB\_B \neq MB\_A$
  12.  $\text{mod}(ia-1, MB\_A) \neq \text{mod}(ib-1, MB\_B)$ .
  13.  $\text{mod}(ia-1, MB\_A) \neq 0$
  14. In the process grid, the process row containing the first row of the submatrix *A* does not contain the first row of the submatrix *B*; that is,  $iarrow \neq ibrow$ , where:
    - $iarrow = \text{mod}(\text{mod}(((ia-1)/MB\_A)+RSRC\_A), p)$
    - $ibrow = \text{mod}(\text{mod}(((ib-1)/MB\_B)+RSRC\_B), p)$

## Stage 6:

1.  $LLD\_A < \max(1, \text{LOCp}(M\_A))$
  2.  $LLD\_B < \max(1, \text{LOCp}(M\_B))$
- Each of the following global input arguments are checked to determine whether its value differs from the value specified on process  $P_{00}$ :
3. *transa* differs.

4.  $n$  differs.
5.  $nrhs$  differs.
6.  $ia$  differs.
7.  $ja$  differs.
8. DTYPE\_A differs.
9. M\_A differs.
10. N\_A differs.
11. MB\_A differs.
12. NB\_A differs.
13. RSRC\_A differs.
14. CSRC\_A differs.
15.  $ib$  differs.
16.  $jb$  differs.
17. DTYPE\_B differs.
18. M\_B differs.
19. N\_B differs.
20. MB\_B differs.
21. NB\_B differs.
22. RSRC\_B differs.
23. CSRC\_B differs.

## Examples

### Example 1

This example solves the real system  $AX = B$  with 5 right-hand sides using a  $2 \times 2$  process grid. The input *ipvt* vector and transformed matrix  $A$  are the output from “Example 1” on page 374.

This example uses a global submatrix  $B$  within a global matrix  $B$  by specifying  $ib = 1$  and  $jb = 2$ .

By specifying  $RSRC\_B = 1$ , the rows of global matrix  $B$  are distributed over the process grid starting in the second row of the process grid. In addition, by specifying  $CSRC\_B = 1$ , the columns of global matrix  $B$  are distributed over the process grid starting in the second column of the process grid.

### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      TRANS A  N  NRHS  A  IA  JA  DESC_A  IPVT  B  IB  JB  DESC_B  INFO
      |    |  |    |  |  |  |  |    |  |  |  |  |  |
CALL PDGETRS( 'N' , 9 , 5 , A , 1 , 1 , DESC_A , IPVT , B , 1 , 2 , DESC_B , INFO )
```

	Desc_A	Desc_B
DTYPE_	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	9	9
N_	9	6
MB_	3	3

	Desc_A	Desc_B
NB_	3	2
RSRC_	1	1
CSRC_	0	1
LLD_	See below <sup>2</sup>	See below <sup>2</sup>

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:  

$$\text{LLD\_A} = \text{MAX}(1, \text{NUMROC}(\text{M\_A}, \text{MB\_A}, \text{MYROW}, \text{RSRC\_A}, \text{NPROW}))$$

$$\text{LLD\_B} = \text{MAX}(1, \text{NUMROC}(\text{M\_B}, \text{MB\_B}, \text{MYROW}, \text{RSRC\_B}, \text{NPROW}))$$

In this example,  $\text{LLD\_A} = \text{LLD\_B} = 3$  on  $P_{00}$  and  $P_{01}$ , and  
 $\text{LLD\_A} = \text{LLD\_B} = 6$  on  $P_{10}$  and  $P_{11}$ .

After the global matrix  $B$  is distributed over the process grid, only a portion of the global data structure is used—that is, global submatrix  $B$ . Following is the global  $9 \times 5$  submatrix  $B$ , starting at row 1 and column 2 in global general  $9 \times 6$  matrix  $B$  with block size  $3 \times 2$ :

B,D	0	1	2
0	$\begin{bmatrix} . & 93.0 \\ . & 84.4 \\ . & 76.6 \end{bmatrix}$	$\begin{bmatrix} 186.0 & 279.0 \\ 168.8 & 253.2 \\ 153.2 & 229.8 \end{bmatrix}$	$\begin{bmatrix} 372.0 & 465.0 \\ 337.6 & 422.0 \\ 306.4 & 383.0 \end{bmatrix}$
1	$\begin{bmatrix} . & 70.0 \\ . & 65.0 \\ . & 62.0 \end{bmatrix}$	$\begin{bmatrix} 140.0 & 210.0 \\ 130.0 & 195.0 \\ 124.0 & 186.0 \end{bmatrix}$	$\begin{bmatrix} 280.0 & 350.0 \\ 260.0 & 325.0 \\ 248.0 & 310.0 \end{bmatrix}$
2	$\begin{bmatrix} . & 61.4 \\ . & 63.6 \\ . & 69.0 \end{bmatrix}$	$\begin{bmatrix} 122.8 & 184.2 \\ 127.2 & 190.8 \\ 138.0 & 207.0 \end{bmatrix}$	$\begin{bmatrix} 245.6 & 307.0 \\ 254.4 & 318.0 \\ 276.0 & 345.0 \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	1	0 2
1	$P_{00}$	$P_{01}$
0 2	$P_{10}$	$P_{11}$

**Note:** The first row of  $B$  begins in the second row of the process grid, and the first column of  $B$  begins in the second column of the process grid.

Local arrays for  $B$ :

p,q	0	1
0	$\begin{bmatrix} 140.0 & 210.0 \\ 130.0 & 195.0 \\ 124.0 & 186.0 \end{bmatrix}$	$\begin{bmatrix} . & 70.0 & 280.0 & 350.0 \\ . & 65.0 & 260.0 & 325.0 \\ . & 62.0 & 248.0 & 310.0 \end{bmatrix}$
1	$\begin{bmatrix} 186.0 & 279.0 \\ 168.8 & 253.2 \\ 153.2 & 229.8 \\ 122.8 & 184.2 \\ 127.2 & 190.8 \\ 138.0 & 207.0 \end{bmatrix}$	$\begin{bmatrix} . & 93.0 & 372.0 & 465.0 \\ . & 84.4 & 337.6 & 422.0 \\ . & 76.6 & 306.4 & 383.0 \\ . & 61.4 & 245.6 & 307.0 \\ . & 63.6 & 254.4 & 318.0 \\ . & 69.0 & 276.0 & 345.0 \end{bmatrix}$

**Output:**

After the global matrix  $B$  is distributed over the process grid, only a portion of the global data structure is used—that is, global submatrix  $B$ . Following is the global  $9 \times 5$  submatrix  $B$ , starting at row 1 and column 2 in global general  $9 \times 6$  matrix  $B$  with block size  $3 \times 2$ :

B,D	0	1	2
0	$\begin{bmatrix} . & 1.0 \\ . & 2.0 \\ . & 3.0 \end{bmatrix}$	$\begin{bmatrix} 2.0 & 3.0 \\ 4.0 & 6.0 \\ 6.0 & 9.0 \end{bmatrix}$	$\begin{bmatrix} 4.0 & 5.0 \\ 8.0 & 10.0 \\ 12.0 & 15.0 \end{bmatrix}$
1	$\begin{bmatrix} . & 4.0 \\ . & 5.0 \\ . & 6.0 \end{bmatrix}$	$\begin{bmatrix} 8.0 & 12.0 \\ 10.0 & 15.0 \\ 12.0 & 18.0 \end{bmatrix}$	$\begin{bmatrix} 16.0 & 20.0 \\ 20.0 & 25.0 \\ 24.0 & 30.0 \end{bmatrix}$
2	$\begin{bmatrix} . & 7.0 \\ . & 8.0 \\ . & 9.0 \end{bmatrix}$	$\begin{bmatrix} 14.0 & 21.0 \\ 16.0 & 24.0 \\ 18.0 & 27.0 \end{bmatrix}$	$\begin{bmatrix} 28.0 & 35.0 \\ 32.0 & 40.0 \\ 36.0 & 45.0 \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	1	0 2
1	$P_{00}$	$P_{01}$
0	$P_{10}$	$P_{11}$
2		

**Note:** The first row of  $B$  begins in the second row of the process grid, and the first column of  $B$  begins in the second column of the process grid.

Local arrays for  $B$ :

p,q	0	1
0	$\begin{bmatrix} 8.0 & 12.0 \\ 10.0 & 15.0 \\ 12.0 & 18.0 \end{bmatrix}$	$\begin{bmatrix} . & 4.0 & 16.0 & 20.0 \\ . & 5.0 & 20.0 & 25.0 \\ . & 6.0 & 24.0 & 30.0 \end{bmatrix}$
1	$\begin{bmatrix} 2.0 & 3.0 \\ 4.0 & 6.0 \\ 6.0 & 9.0 \\ 14.0 & 21.0 \\ 16.0 & 24.0 \\ 18.0 & 27.0 \end{bmatrix}$	$\begin{bmatrix} . & 1.0 & 4.0 & 5.0 \\ . & 2.0 & 8.0 & 10.0 \\ . & 3.0 & 12.0 & 15.0 \\ . & 7.0 & 28.0 & 35.0 \\ . & 8.0 & 32.0 & 40.0 \\ . & 9.0 & 36.0 & 45.0 \end{bmatrix}$

The value of *info* is 0 on all processes.

**Example 2**

This example solves the complex system  $AX = B$  with 5 right-hand sides using a  $2 \times 2$  process grid. The input *ipvt* vector and transformed matrix  $A$  are the output from “Example 2” on page 377.

This example uses a global submatrix  $B$  within a global matrix  $B$  by specifying  $ib = 1$  and  $jb = 2$ .

By specifying  $RSRC\_B = 1$ , the rows of global matrix  $B$  are distributed over the process grid starting in the second row of the process grid. In addition, by specifying  $CSRC\_B = 1$ , the columns of global matrix  $B$  are distributed over the process grid starting in the second column of the process grid.

## PDGETRS and PZGETRS

### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      TRANSA N  NRHS  A  IA  JA  DESC_A  IPVT  B  IB  JB  DESC_B  INFO
      |      |  |    |  |  |  |      |  |  |  |  |  |
CALL PZGETRS( 'N' , 9 , 5 , A , 1 , 1 , DESC_A , IPVT , B , 1 , 2 , DESC_B , INFO )
```

	Desc_A	Desc_B
DTYPE_	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	9	9
N_	9	6
MB_	3	3
NB_	3	2
RSRC_	1	1
CSRC_	0	1
LLD_	See below <sup>2</sup>	See below <sup>2</sup>

#### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.

2. Each process should set the LLD\_ as follows:

```
LLD_A = MAX(1, NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_B = MAX(1, NUMROC(M_B, MB_B, MYROW, RSRC_B, NPROW))
```

In this example, LLD\_A = LLD\_B = 3 on P<sub>00</sub> and P<sub>01</sub>, and

LLD\_A = LLD\_B = 6 on P<sub>10</sub> and P<sub>11</sub>.

After the global matrix **B** is distributed over the process grid, only a portion of the global data structure is used—that is, global submatrix **B**. Following is the global 9 × 5 submatrix **B**, starting at row 1 and column 2 in global general 9 × 6 matrix **B** with block size 3 × 2:

B,D	0	1	2
0	. (193.0, -10.6) . (173.8, -9.4) . (156.2, -5.4)	(200.0, 21.8) (207.0, 54.2) (178.8, 20.2) (183.8, 49.8) (159.2, 22.2) (162.2, 49.8)	(214.0, 86.6) (221.0, 119.0) (188.8, 79.4) (193.8, 109.0) (165.2, 77.4) (168.2, 105.0)
1	. (141.0, 1.4) . (129.0, 11.0) . (121.0, 23.4)	(142.0, 27.8) (143.0, 54.2) (128.0, 37.0) (127.0, 63.0) (118.0, 49.8) (115.0, 76.2)	(144.0, 80.6) (145.0, 107.0) (126.0, 89.0) (125.0, 115.0) (112.0, 102.6) (109.0, 129.0)
2	. (117.8, 38.6) . (120.2, 56.6) . (129.0, 77.4)	(112.8, 66.2) (107.8, 93.8) (113.2, 86.2) (106.2, 115.8) (120.0, 109.8) (111.0, 142.2)	(102.8, 121.4) (97.8, 149.0) (99.2, 145.4) (92.2, 175.0) (102.0, 174.6) (93.0, 207.0)

The following is the 2 × 2 process grid:



B,D	1	0 2
1	P <sub>00</sub>	P <sub>01</sub>
0	P <sub>10</sub>	P <sub>11</sub>
2		

**Note:** The first row of  $B$  begins in the second row of the process grid, and the first column of  $B$  begins in the second column of the process grid.

Local arrays for  $B$ :

p,q	0	1
0	(142.0, 27.8) (143.0, 54.2) (128.0, 37.0) (127.0, 63.0) (118.0, 49.8) (115.0, 76.2)	. (141.0, 1.4) (144.0, 80.6) (145.0, 107.0) . (129.0, 11.0) (126.0, 89.0) (125.0, 115.0) . (121.0, 23.4) (112.0, 102.6) (109.0, 129.0)
1	(200.0, 21.8) (207.0, 54.2) (178.8, 20.2) (183.8, 49.8) (159.2, 22.2) (162.2, 49.8) (112.8, 66.2) (107.8, 93.8) (113.2, 86.2) (106.2, 115.8) (120.0, 109.8) (111.0, 142.2)	. (193.0, -10.6) (214.0, 86.6) (221.0, 119.0) . (173.8, -9.4) (188.8, 79.4) (193.8, 109.0) . (156.2, -5.4) (165.2, 77.4) (168.2, 105.0) . (117.8, 38.6) (102.8, 121.4) (97.8, 149.0) . (120.2, 56.6) (99.2, 145.4) (92.2, 175.0) . (129.0, 77.4) (102.0, 174.6) (93.0, 207.0)

**Output:**

After the global matrix  $B$  is distributed over the process grid, only a portion of the global data structure is used—that is, global submatrix  $B$ . Following is the global  $9 \times 5$  submatrix  $B$ , starting at row 1 and column 2 in global general  $9 \times 6$  matrix  $B$  with block size  $3 \times 2$ :

B,D	0	1	2
0	. (1.0, 1.0) . (2.0, 1.0) . (3.0, 1.0)	(1.0, 2.0) (1.0, 3.0) (2.0, 2.0) (2.0, 3.0) (3.0, 2.0) (3.0, 3.0)	(1.0, 4.0) (1.0, 5.0) (2.0, 4.0) (2.0, 5.0) (3.0, 4.0) (3.0, 5.0)
1	. (4.0, 1.0) . (5.0, 1.0) . (6.0, 1.0)	(4.0, 2.0) (4.0, 3.0) (5.0, 2.0) (5.0, 3.0) (6.0, 2.0) (6.0, 3.0)	(4.0, 4.0) (4.0, 5.0) (5.0, 4.0) (5.0, 5.0) (6.0, 4.0) (6.0, 5.0)
2	. (7.0, 1.0) . (8.0, 1.0) . (9.0, 1.0)	(7.0, 2.0) (7.0, 3.0) (8.0, 2.0) (8.0, 3.0) (9.0, 2.0) (9.0, 3.0)	(7.0, 4.0) (7.0, 5.0) (8.0, 4.0) (8.0, 5.0) (9.0, 4.0) (9.0, 5.0)

The following is the  $2 \times 2$  process grid:

B,D	1	0 2
1	P <sub>00</sub>	P <sub>01</sub>
0	P <sub>10</sub>	P <sub>11</sub>
2		

**Note:** The first row of  $B$  begins in the second row of the process grid, and the first column of  $B$  begins in the second column of the process grid.

Local arrays for  $B$ :

## PDGETRI and PZGETRI

p,q	0		1			
0	(3.0, 2.0)	(3.0, 3.0)	.	(3.0, 1.0)	(3.0, 4.0)	(3.0, 5.0)
	(4.0, 2.0)	(4.0, 3.0)	.	(4.0, 1.0)	(4.0, 4.0)	(4.0, 5.0)
	(5.0, 2.0)	(5.0, 3.0)	.	(5.0, 1.0)	(5.0, 4.0)	(5.0, 5.0)
1	(1.0, 2.0)	(1.0, 3.0)	.	(1.0, 1.0)	(1.0, 4.0)	(1.0, 5.0)
	(2.0, 2.0)	(2.0, 3.0)	.	(2.0, 1.0)	(2.0, 4.0)	(2.0, 5.0)
	(3.0, 2.0)	(3.0, 3.0)	.	(3.0, 1.0)	(3.0, 4.0)	(3.0, 5.0)
	(7.0, 2.0)	(7.0, 3.0)	.	(7.0, 1.0)	(7.0, 4.0)	(7.0, 5.0)
	(8.0, 2.0)	(8.0, 3.0)	.	(8.0, 1.0)	(8.0, 4.0)	(8.0, 5.0)
	(9.0, 2.0)	(9.0, 3.0)	.	(9.0, 1.0)	(9.0, 4.0)	(9.0, 5.0)

The value of *info* is 0 on all processes.

## PDGETRI and PZGETRI — General Matrix Inverse

### Purpose

PDGETRI and PZGETRI compute the inverse of general matrix  $A$ . These subroutines use the results of the factorization of matrix  $A$ , produced by a preceding call to PDGETRF or PZGETRF, respectively. For details on the factorization, see “PDGETRF and PZGETRF — General Matrix Factorization” on page 370.

If  $n = 0$ , no computation is performed and the subroutine returns after doing some parameter checking.

Table 73. Data Types

$A$ , $work$	$ipvt$ , $iwork$	Subroutine
Long-precision real	Integer	PDGETRI
Long-precision complex	Integer	PZGETRI

### Syntax

<b>Fortran</b>	CALL PDGETRI   PZGETRI ( $n$ , $a$ , $ia$ , $ja$ , $desc\_a$ , $ipvt$ , $work$ , $lwork$ , $iwork$ , $liwork$ , $info$ )
<b>C and C++</b>	pdgetri   pzgetri ( $n$ , $a$ , $ia$ , $ja$ , $desc\_a$ , $ipvt$ , $work$ , $lwork$ , $iwork$ , $liwork$ , $info$ );

### On Entry

$n$  is the order of the factored submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

$a$  is the local part of the global general matrix  $A$ , containing the factorization of matrix  $A$  produced by a preceding call to PDGETRF or PZGETRF, respectively. This identifies the **first element** of the local array  $A$ . This subroutine computes the location of the first element of the local subarray used, based on  $ia$ ,  $ja$ ,  $desc\_a$ ,  $p$ ,  $q$ ,  $myrow$ , and  $mycol$ ; therefore, the leading  $LOCp(ia+n-1)$  by  $LOCq(ja+n-1)$  part of the local array  $A$  must contain the local pieces of the leading  $ia+n-1$  by  $ja+n-1$  part of the global matrix.

Scope: **local**

Specified as: an  $LLD\_A$  by (at least)  $LOCq(N\_A)$  array, containing numbers of the data type indicated in Table 73. Details about the square block-cyclic data distribution of global matrix  $A$  are stored in  $desc\_a$ .

$ia$  is the row index of the global matrix  $A$ , identifying the first row of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq M\_A$  and  $ia+n-1 \leq M\_A$ .

$ja$  is the column index of the global matrix  $A$ , identifying the first column of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$  and  $ja+n-1 \leq N\_A$ .

$desc\_a$  is the array descriptor for global matrix  $A$ , described in the following table:

## PDGETRI and PZGETRI

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_A < p$	Global
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < q$	Global
9	LLD_A	The leading dimension of the local array	$LLD\_A \geq \max(1, LOCp(M\_A))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

*ipvt* is the local part of the global vector *ipvt*, containing the pivoting indices produced on a preceding call to PDGETRF or PZGETRF, respectively. This identifies the **first element** of the local array IPVT. This subroutine computes the location of the first element of the local subarray used, based on *ia*, *desc\_a*, *p*, and *myrow*; therefore, the leading  $LOCp(ia+n-1)$  part of the local array IPVT must contain the local pieces of the leading  $ia+n-1$  part of the global vector.

A copy of the vector *ipvt*, with a block size of MB\_A and global index *ia*, is contained in each column of the process grid. The process row over which the first row of *ipvt* is distributed is RSRC\_A.

Scope: **local**

Specified as: an array of (at least) length  $LOCp(ia+n-1)$ , containing fullword integers, where  $ia \leq$  (pivoting index values)  $\leq ia+n-1$ . Details about the block-cyclic data distribution of global vector *ipvt* are stored in *desc\_a*.

*work* has the following meaning:

If  $lwork = 0$ , *work* is ignored.

If  $lwork \neq 0$ , *work* is a work area used by this subroutine, where:

- If  $lwork \neq -1$ , then its size is (at least) of length *lwork*.
- If  $lwork = -1$ , then its size is (at least) of length 1.

Scope: **local**

Specified as: an area of storage containing numbers of data type indicated in Table 73 on page 393.

*lwork* is the number of elements in array WORK.

Scope:

- If  $lwork \geq 0$ , *lwork* is **local**.
- If  $lwork = -1$ , *lwork* is **global**.

Specified as: a fullword integer; where:

- If  $lwork = 0$ , PDGETRI and PZGETRI dynamically allocate the work area used by the subroutine. The work area is deallocated before control is returned to the calling program. This option is an extension to the ScaLAPACK standard.
- If  $lwork = -1$ , PDGETRI and PZGETRI perform a work area query and return the minimum size of *work* in  $work_1$ . No computation is performed and the subroutine returns after error checking is complete.
- Otherwise, it must have the following value:

$$lwork = \text{NUMROC}(n + iroff, MB\_A, myrow, iarow, nprow) * NB\_A$$

where:

- $iroff = \text{mod}(ia-1, MB\_A)$
- $iarow = \text{mod}(RSRC\_A + (ia-1)/MB\_A, nprow)$

*iwork* has the following meaning:

If  $liwork = 0$ , *iwork* is ignored.

If  $liwork \neq 0$ , *iwork* is a work area used by this subroutine, where:

- If  $liwork \neq -1$ , then its size is (at least) of length *liwork*.
- If  $liwork = -1$ , then its size is (at least) of length 1.

Scope: **local**

Specified as: an area of storage containing fullword integers.

*liwork* is the number of elements in array IWORK.

Scope:

- If  $liwork \geq 0$ , *liwork* is **local**.
- If  $liwork = -1$ , *liwork* is **global**.

Specified as: a fullword integer; where:

- If  $liwork = 0$ , PDGETRI and PZGETRI dynamically allocate the work area used by the subroutine. The work area is deallocated before control is returned to the calling program. This option is an extension to the ScaLAPACK standard.
- If  $liwork = -1$ , PDGETRI and PZGETRI perform a work area query and return the minimum size of *iwork* in  $iwork_1$ . No computation is performed and the subroutine returns after error checking is complete.
- Otherwise, use the following rules to determine the value to specify:

$$\text{If } nprow = npcol, \text{ then } liwork = nq + NB\_A$$

$$\text{If } nprow \neq npcol, \text{ then } liwork = nq + \max(liwork1, liwork2, NB\_A)$$

where:

- $liwork1 = MB\_A * \text{iceil}(\text{iceil}(mp, MB\_A), lcm/nprow)$
- $liwork2 = NB\_A * \text{iceil}(\text{iceil}(mq, NB\_A), lcm/npcol)$
- $mp = \text{NUMROC}(M\_A, MB\_A, myrow, RSRC\_A, nprow)$

- $mq = \text{NUMROC}(M\_A, MB\_A, mycol, CSRC\_A, npcol)$
- $nq = \text{NUMROC}(N\_A, NB\_A, mycol, CSRC\_A, npcol)$
- $lcm = \text{ilcm}(nprow, npcol)$

*info* See On Return.

### On Return

*a* is the updated local part of the global general matrix *A*, containing the inverse of matrix *A*.

Scope: **local**

Returned as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 73 on page 393.

*work* is the work area used by this subroutine if  $lwork \neq 0$ , where:

If  $lwork \neq 0$  and  $lwork \neq -1$ , its size is (at least) of length *lwork*.

If  $lwork = -1$ , its size is (at least) of length 1.

Scope: **local**

Returned as: an area of storage, containing numbers of the data type indicated in Table 73 on page 393, where:

If  $lwork \geq 1$  or  $lwork = -1$ , then  $work_1$  is set to the minimum *lwork* value needed. Except for  $work_1$ , the contents of *work* are overwritten on return.

*iwork* is the work area used by this subroutine if  $liwork \neq 0$ , where:

If  $liwork \neq 0$  and  $liwork \neq -1$ , then its size is (at least) of length *liwork*.

If  $liwork = -1$ , then its size is (at least) of length 1.

Scope: **local**

Returned as: an area of storage, where:

If  $liwork \geq 1$  or  $liwork = -1$ , then  $iwork_1$  is set to the minimum *liwork* value and contains numbers of the data type indicated in Table 73 on page 393. Except for  $iwork_1$ , the contents of *iwork* are overwritten on return.

*info* has the following meaning:

If  $info = 0$ , global submatrix *A* is not singular, and the inverse completed normally.

If  $info > 0$ , global submatrix *A* is singular and the inverse could not be computed. For  $info = k$ , the corresponding diagonal element of  $U_{k, k}$  is exactly zero.

Scope: **global**

Returned as: a fullword integer;  $info \geq 0$ .

## Notes and Coding Rules

1. In your C program, argument *info* must be passed by reference.
2. The matrix and vector must have no common elements; otherwise, results are unpredictable.
3. The scalar data specified for input argument *n* must be the same for both PDGETRF/PZGETRF and PDGETRI/PZGETRI. In addition, the scalar data specified for input argument *m* in PDGETRF/PZGETRF **must be the same** as input argument *n* in both PDGETRF/PZGETRF and PDGETRI/PZGETRI.

If, however, you do **not** plan to call PDGETRI/PZGETRI after calling PDGETRF/PZGETRF, then input arguments  $m$  and  $n$  in PDGETRF/PZGETRF do not need to be equal.

4. The global submatrices for  $A$  and *ipvt* input to PDGETRI/PZGETRI must be the same as for the corresponding output arguments for PDGETRF/PZGETRF; and thus, the scalar data specified for  $ia$ ,  $ja$ , and the contents of *desc\_a* must also be the same.
5. The NUMROC utility subroutine can be used to determine the values of LOCp(M\_) and LOCq(N\_) used in the argument descriptions above. For details, see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28 and “NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process” on page 868.
6. For suggested block sizes, see “Coding Tips for Optimizing Parallel Performance” on page 77.
7. On both input and output, matrix  $A$  conforms to ScaLAPACK format.
8. The global general matrix  $A$  must be distributed using a square block-cyclic distribution; that is,  $MB\_A = NB\_A$ .
9. The global general matrix  $A$  must be aligned on a block row boundary; that is,  $ia-1$  must be a multiple of  $MB\_A$ .
10. The block row offset of  $A$  must be equal to the block column offset of  $A$ ; that is,  $\text{mod}(ia-1, MB\_A) = \text{mod}(ja-1, NB\_A)$ .
11. There is no array descriptor for *ipvt*. It is a column-distributed vector with block size  $MB\_A$ , local arrays of dimension LOCp( $ia+n-1$ ) by 1, and global index  $ia$ . A copy of this vector exists on each column of the process grid, and the process row over which the first column of *ipvt* is distributed is RSRC\_A.

## Error Conditions

### Computational Errors

Matrix  $A$  is a singular matrix. For details, see the description of the *info* argument.

### Resource Errors

Unable to allocate work space

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. DTYPE\_A is invalid.

#### Stage 2:

1. CTXT\_A is invalid.

#### Stage 3:

1. This subroutine was called from outside the process grid.

#### Stage 4:

1.  $n < 0$
2.  $M\_A < 0$  and  $n = 0$ ;  $M\_A < 1$  otherwise
3.  $N\_A < 0$  and  $n = 0$ ;  $N\_A < 1$  otherwise
4.  $ia < 1$
5.  $ja < 1$
6.  $MB\_A < 1$
7.  $NB\_A < 1$

8.  $\text{RSRC\_A} < 0$  or  $\text{RSRC\_A} \geq p$
9.  $\text{CSRC\_A} < 0$  or  $\text{CSRC\_A} \geq q$

**Stage 5:** If  $n \neq 0$ :

1.  $ia > M\_A$
2.  $ja > N\_A$
3.  $ia+n-1 > M\_A$
4.  $ja+n-1 > N\_A$

In all cases:

5.  $MB\_A \neq NB\_A$
6.  $\text{mod}(ia-1, MB\_A) \neq \text{mod}(ja-1, NB\_A)$
7.  $\text{mod}(ia-1, MB\_A) \neq 0$

**Stage 6:**

1.  $LLD\_A < \max(1, \text{LOCp}(M\_A))$
2.  $lwork \neq 0$ ,  $lwork \neq -1$ , and  $lwork < (\text{minimum value})$ . (For the minimum value, see the *lwork* argument description.)
3.  $liwork \neq 0$ ,  $liwork \neq -1$ , and  $liwork < (\text{minimum value})$ . (For the minimum value, see the *liwork* argument description.)

**Stage 7:** Each of the following global input arguments are checked to determine whether its value differs from the value specified on process  $P_{00}$ :

1.  $n$  differs.
2.  $ia$  differs.
3.  $ja$  differs.
4.  $\text{DTYPE\_A}$  differs.
5.  $M\_A$  differs.
6.  $N\_A$  differs.
7.  $MB\_A$  differs.
8.  $NB\_A$  differs.
9.  $\text{RSRC\_A}$  differs.
10.  $\text{CSRC\_A}$  differs.

Also:

11.  $lwork = -1$  on a subset of processes.
12.  $liwork = -1$  on a subset of processes.

## Examples

### Example 1

This example computes the inverse of a real matrix using the **LU** factorization computed by PDGETRF. The input *ipvt* vector and transformed matrix *A* are the output from PDGETRF, "Example 1" on page 374.

**Note:**

Because  $lwork = 0$  and  $liwork = 0$ , PDGETRI dynamically allocates the work areas used by this subroutine.



## Call Statements and Input:

```

ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      N   A   IA  JA  DESC_A  IPVT  WORK  LWORK  IWORK  LIWORK  INFO
      |   |   |   |   |       |   |   |   |   |   |   |
CALL PDGETRI( 9 , A , 1 , 1 , DESC_A , IPVT, WORK , 0 , IWORK , 0 , INFO )

```

	Desc_A
DTYPE_	1
CTXT_	<i>icontxt</i> <sup>1</sup>
M_	9
N_	9
MB_	3
NB_	3
RSRC_	1
CSRC_	0
LLD_	See below <sup>2</sup>

## Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:  

$$\text{LLD\_A} = \text{MAX}(1, \text{NUMROC}(\text{M\_A}, \text{MB\_A}, \text{MYROW}, \text{RSRC\_A}, \text{NPROW}))$$
In this example,  $\text{LLD\_A} = 3$  on  $P_{00}$  and  $P_{01}$ , and  $\text{LLD\_A} = 6$  on  $P_{10}$  and  $P_{11}$ .

## Output:

Global general  $9 \times 9$  inverted matrix  $A$  with block size  $3 \times 3$ :

B,D	0	1	2
0	$\begin{bmatrix} -2.4 & 2.5 & 0.0 \\ 2.5 & -5.0 & 2.5 \\ 0.0 & 2.5 & -5.0 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 2.5 & 0.0 & 0.0 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 0.0 & 0.1 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}$
1	$\begin{bmatrix} 0.0 & 0.0 & 2.5 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}$	$\begin{bmatrix} -5.0 & 2.5 & 0.0 \\ 2.5 & -5.0 & 2.5 \\ 0.0 & 2.5 & -5.0 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 2.5 & 0.0 & 0.0 \end{bmatrix}$
2	$\begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.1 & 0.0 & 0.0 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 0.0 & 2.5 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}$	$\begin{bmatrix} -5.0 & 2.5 & 0.0 \\ 2.5 & -5.0 & 2.5 \\ 0.0 & 2.5 & -2.4 \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
1	$P_{00}$	$P_{01}$
0	$P_{10}$	$P_{11}$
2		

**Note:** The first row of *A* begins in the second row of the process grid.

Local arrays for *A*:

p,q	0						1		
0	0.0	0.0	2.5	0.0	0.0	0.0	-5.0	2.5	0.0
	0.0	0.0	0.0	0.0	0.0	0.0	2.5	-5.0	2.5
	0.0	0.0	0.0	2.5	0.0	0.0	0.0	2.5	-5.0
1	-2.4	2.5	0.0	0.0	0.0	0.1	0.0	0.0	0.0
	2.5	-5.0	2.5	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	2.5	-5.0	0.0	0.0	0.0	2.5	0.0	0.0
	0.0	0.0	0.0	-5.0	2.5	0.0	0.0	0.0	2.5
	0.0	0.0	0.0	2.5	-5.0	2.5	0.0	0.0	0.0
	0.1	0.0	0.0	0.0	2.5	-2.4	0.0	0.0	0.0

The value of *info* is 0 on all processes.

## Example 2

This example computes the inverse of a complex matrix using the *LU* factorization computed by PZGETRF. The input *ipvt* vector and transformed matrix *A* are the output from PZGETRF, "Example 2" on page 377.

**Note:**

Because *lwork* = 0 and *liwork* = 0, PZGETRI dynamically allocates the work areas used by this subroutine.

### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      N   A   IA  JA  DESC_A  IPVT  WORK  LWORK  IWORK  LIWORK  INFO
      |   |   |   |   |      |   |   |   |   |   |   |
CALL PZGETRI( 9 , A , 1 , 1 , DESC_A , IPVT , WORK , 0 , IWORK , 0 , INFO )
```

	Desc_A
DTYPE_	1
CTXT_	<i>icontxt</i> <sup>1</sup>
M_	9
N_	9
MB_	3
NB_	3
RSRC_	1
CSRC_	0
LLD_A	See below <sup>2</sup>

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:  
`LLD_A = MAX(1, NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))`

In this example,  $LLD\_A = 3$  on  $P_{00}$  and  $P_{01}$ , and  $LLD\_A = 6$  on  $P_{10}$  and  $P_{11}$ .

**Output:**

Global general  $9 \times 9$  inverted complex matrix  $A$  with block size  $3 \times 3$ :

B,D	0	1	2
0	$\begin{pmatrix} (-.17, -.42) & (-.12, .13) & (-.06, .15) \\ (.18, .43) & (-.04, -.55) & (-.06, -.03) \\ (.01, .00) & (.18, .43) & (-.04, -.55) \end{pmatrix}$	$\begin{pmatrix} (.00, .15) & (.05, .13) & (.09, .09) \\ (-.06, -.01) & (-.05, .01) & (-.04, .03) \\ (-.06, -.03) & (-.06, -.01) & (-.05, .01) \end{pmatrix}$	$\begin{pmatrix} (.11, .05) & (.11, .00) & (.04, -.28) \\ (-.03, .04) & (-.01, .04) & (.11, .00) \\ (-.04, .03) & (-.03, .04) & (.11, .05) \end{pmatrix}$
1	$\begin{pmatrix} (.01, .00) & (.01, .00) & (.18, .43) \\ (.00, .01) & (.01, .00) & (.01, .00) \\ (.00, .01) & (.00, .01) & (.01, .00) \end{pmatrix}$	$\begin{pmatrix} (-.04, -.55) & (-.06, -.03) & (-.06, -.01) \\ (.18, .43) & (-.04, -.55) & (-.06, -.03) \\ (.01, .00) & (.18, .43) & (-.04, -.55) \end{pmatrix}$	$\begin{pmatrix} (-.05, .01) & (-.04, .03) & (.09, .09) \\ (-.06, -.01) & (-.05, .01) & (.05, .13) \\ (-.06, -.03) & (-.06, -.01) & (.00, .15) \end{pmatrix}$
2	$\begin{pmatrix} (.00, .01) & (.00, .01) & (.00, .01) \\ (.00, .01) & (.00, .01) & (.00, .01) \\ (-.01, .01) & (.00, .01) & (.00, .01) \end{pmatrix}$	$\begin{pmatrix} (.01, .00) & (.01, .00) & (.18, .43) \\ (.00, .01) & (.01, .00) & (.01, .00) \\ (.00, .01) & (.00, .01) & (.01, .00) \end{pmatrix}$	$\begin{pmatrix} (-.04, -.55) & (-.06, -.03) & (-.06, .15) \\ (.18, .43) & (-.04, -.55) & (-.12, .13) \\ (.01, .00) & (.18, .43) & (-.17, -.42) \end{pmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
1	$P_{00}$	$P_{01}$
0	$P_{10}$	$P_{11}$
2		

**Note:** The first row of  $A$  begins in the second row of the process grid.

Local arrays for  $A$ :

p,q	0	1
0	$\begin{pmatrix} (.01, .00) & (.01, .00) & (.18, .43) & (-.05, .01) & (-.04, .03) & (.09, .09) \\ (.00, .01) & (.01, .00) & (.01, .00) & (-.06, -.01) & (-.05, .01) & (.05, .13) \\ (.00, .01) & (.00, .01) & (.01, .00) & (-.06, -.03) & (-.06, -.01) & (.00, .15) \end{pmatrix}$	$\begin{pmatrix} (-.04, -.55) & (-.06, -.03) & (-.06, -.01) \\ (.18, .43) & (-.04, -.55) & (-.06, -.03) \\ (.01, .00) & (.18, .43) & (-.04, -.55) \end{pmatrix}$
1	$\begin{pmatrix} (-.17, -.42) & (-.12, .13) & (-.06, .15) & (.11, .05) & (.11, .00) & (.04, -.28) \\ (.18, .43) & (-.04, -.55) & (-.06, -.03) & (-.03, .04) & (-.01, .04) & (.11, .00) \\ (.01, .00) & (.18, .43) & (-.04, -.55) & (-.04, .03) & (-.03, .04) & (.11, .05) \\ (.00, .01) & (.00, .01) & (.00, .01) & (-.04, -.55) & (-.06, -.03) & (-.06, .15) \\ (.00, .01) & (.00, .01) & (.00, .01) & (.18, .43) & (-.04, -.55) & (-.12, .13) \\ (-.01, .01) & (.00, .01) & (.00, .01) & (.01, .00) & (.18, .43) & (-.17, -.42) \end{pmatrix}$	$\begin{pmatrix} (.00, .15) & (.05, .13) & (.09, .09) \\ (-.06, -.01) & (-.05, .01) & (-.04, .03) \\ (-.06, -.03) & (-.06, -.01) & (-.05, .01) \\ (.01, .00) & (.01, .00) & (.18, .43) \\ (.00, .01) & (.01, .00) & (.01, .00) \\ (.00, .01) & (.00, .01) & (.01, .00) \end{pmatrix}$

The value of *info* is 0 on all processes.

## PDGECON and PZGECON — Estimate the Reciprocal of the Condition Number of a General Matrix

### Purpose

PDGECON and PZGECON estimate the reciprocal of the condition number of general matrix  $A$ . These subroutines use the results of the factorization of matrix  $A$ , produced by a preceding call to PDGETRF or PZGETRF, respectively. For details on the factorization, see “PDGETRF and PZGETRF — General Matrix Factorization” on page 370.

If  $n = 0$ , the subroutines return with  $rcond = 1.0$

See references [17], [19], [23], [38], and [39].

Table 74. Data Types

$A, work$	$anorm, rcond$	$iwork$	$rwork$	Subroutine
Long-precision real	Long-precision real	Integer		PDGECON
Long-precision complex	Long-precision real		Long-precision real	PZGECON

### Syntax

Fortran	CALL PDGECON ( $norm, n, a, ia, ja, desc\_a, anorm, rcond, work, lwork, iwork, liwork, info$ ) CALL PZGECON ( $norm, n, a, ia, ja, desc\_a, anorm, rcond, work, lwork, rwork, lrwork, info$ )
C and C++	pdgecon ( $norm, n, a, ia, ja, desc\_a, anorm, rcond, work, lwork, iwork, liwork, info$ ); pzgecon ( $norm, n, a, ia, ja, desc\_a, anorm, rcond, work, lwork, rwork, lrwork, info$ );

### On Entry

$norm$  specifies whether the estimate of the condition number is computed using the one norm or the infinity norm; where:

If  $norm = 'O'$  or  $'1'$ , the one norm is used in the computation.

If  $norm = 'I'$ , the infinity norm is used in the computation.

Scope: **global**

Specified as: a single character;  $norm = 'O', '1',$  or  $'I'$ .

$n$  is the order of the factored submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

$a$  is the local part of the global general matrix  $A$ , containing the factorization of matrix  $A$  produced by a preceding call to PDGETRF or PZGETRF, respectively. This identifies the **first element** of the local array  $A$ . This subroutine computes the location of the first element of the local subarray used, based on  $ia, ja, desc\_a, p, q, myrow$ , and  $mycol$ ; therefore, the leading  $LOCp(ia+n-1)$  by  $LOCq(ja+n-1)$  part of the local array  $A$  must contain the local pieces of the leading  $ia+n-1$  by  $ja+n-1$  part of the global matrix.

Scope: **local**

Specified as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 74 on page 402. Details about the square block-cyclic data distribution of global matrix *A* are stored in *desc\_a*.

*ia* is the row index of the global matrix *A*, identifying the first row of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq M\_A$  and  $ia+n-1 \leq M\_A$ .

*ja* is the column index of the global matrix *A*, identifying the first column of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$  and  $ja+n-1 \leq N\_A$ .

*desc\_a* is the array descriptor for global matrix *A*, described in the following table:

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_A < p$	Global
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < q$	Global
9	LLD_A	The leading dimension of the local array	$LLD\_A \geq \max(1, LOCp(M\_A))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

*anorm* has the following meaning:

If *norm* = 'O' or '1', then *anorm* is the one norm of the original matrix.

If *norm* = 'I', then *anorm* is the infinity norm of the original matrix.

**Note:** You may obtain the value of *anorm* by a preceding call to PDLANGE or PZLANGE, respectively. Refer to "PDLANGE and PZLANGE — General Matrix Norm" on page 872.

Scope: **global**

Specified as: long-precision real number  $\geq 0.0$

*rcond* See On Return.

*work* has the following meaning:

If  $lwork = 0$ , *work* is ignored.

If  $lwork \neq 0$ , *work* is a work area used by this subroutine, where:

- If  $lwork \neq -1$ , then its size is (at least) of length *lwork*.
- If  $lwork = -1$ , then its size is (at least) of length 1.

Scope: **local**

Specified as: an area of storage containing numbers of data type indicated in Table 74 on page 402.

*lwork* is the number of elements in array WORK.

Scope:

- If  $lwork \geq 0$ , *lwork* is **local**.
- If  $lwork = -1$ , *lwork* is **global**.

Specified as: a fullword integer; where:

- If  $lwork = 0$ , PDGECON and PZGECON dynamically allocate the work area used by the subroutine. The work area is deallocated before control is returned to the calling program. This option is an extension to the ScaLAPACK standard.

- If  $lwork = -1$ , PDGECON and PZGECON perform a work area query and return the minimum size of *work* in *work*<sub>1</sub>. No computation is performed and the subroutine returns after error checking is complete.

- Otherwise, use the following rules to determine the value to specify:

For PDGECON,  $lwork \geq 2np0 + 2nq0 + \max(2, \max(nb(\max(1, \text{iceil}(nprow-1, npcol))), nq0 + nb(\max(1, \text{iceil}(npcol-1, nprow))))$

For PZGECON,  $lwork \geq 2np0 + \max(2, \max(nb(\max(1, \text{iceil}(nprow-1, npcol))), nq0 + nb(\max(1, \text{iceil}(npcol-1, nprow))))$

where:

- $mb = MB\_A$
- $nb = NB\_A$
- $iroff = \text{mod}(ia-1, mb)$
- $icoff = \text{mod}(ja-1, nb)$
- $iarow = \text{mod}(RSRC\_A + (ia-1)/mb, nprow)$
- $iacol = \text{mod}(CSRC\_A + (ja-1)/nb, npcol)$
- $np0 = \text{NUMROC}(n+iroff, mb, myrow, iarow, nprow)$
- $nq0 = \text{NUMROC}(n+icoff, nb, mycol, iacol, npcol)$

*iwork* has the following meaning:

If  $liwork = 0$ , *iwork* is ignored.

If  $liwork \neq 0$ , *iwork* is a work area used by this subroutine, where:

- If  $liwork \neq -1$ , then its size is (at least) of length *liwork*.
- If  $liwork = -1$ , then its size is (at least) of length 1.

Scope: **local**

Specified as: an area of storage containing fullword integers.

*liwork* is the number of elements in array iwork.

Scope:

- If  $liwork \geq 0$ , *liwork* is **local**.

- If  $liwork = -1$ ,  $liwork$  is **global**.

Specified as: a fullword integer; where:

- If  $liwork = 0$ , PDGECON dynamically allocates the work area used by the subroutine. The work area is deallocated before control is returned to the calling program. This option is an extension to the ScaLAPACK standard.
- If  $liwork = -1$ , PDGECON performs a work area query and return the minimum size of  $iwork$  in  $iwork_1$ . No computation is performed and the subroutine returns after error checking is complete.
- Otherwise, use the following rules to determine the value to specify:

$$liwork \geq np0$$

where:

- $mb = MB\_A$
- $irow = \text{mod}(ia-1, mb)$
- $iarow = \text{mod}(RSRC\_A + (ia-1)/mb, nprow)$
- $np0 = \text{NUMROC}(n+irow, mb, myrow, iarow, nprow)$

$rwork$  has the following meaning:

If  $lrwork = 0$ ,  $rwork$  is ignored.

If  $lrwork \neq 0$ ,  $rwork$  is a work area used by this subroutine, where:

- If  $lrwork \neq -1$ , then its size is (at least) of length  $lrwork$ .
- If  $lrwork = -1$ , then its size is (at least) of length 1.

Scope: **local**

Specified as: an area of storage containing long-precision real numbers.

$lrwork$  is the number of elements in array  $rwork$ .

Scope:

- If  $lrwork \geq 0$ ,  $lrwork$  is **local**.
- If  $lrwork = -1$ ,  $lrwork$  is **global**.

Specified as: a fullword integer; where:

- If  $lrwork = 0$ , PZGECON dynamically allocates the work area used by the subroutine. The work area is deallocated before control is returned to the calling program. This option is an extension to the ScaLAPACK standard.
- If  $lrwork = -1$ , PZGECON performs a work area query and return the minimum size of  $rwork$  in  $rwork_1$ . No computation is performed and the subroutine returns after error checking is complete.
- Otherwise, use the following rules to determine the value to specify:

$$lrwork \geq 2nq0$$

where:

- $nb = NB\_A$
- $icoff = \text{mod}(ja-1, nb)$
- $iacol = \text{mod}(CSRC\_A + (ja-1)/nb, npcol)$
- $nq0 = \text{NUMROC}(n+icoff, nb, mycol, iacol, npcol)$

*info* See On Return.

## On Return

$rcond$  has the following meaning:

If  $info = 0$ , an estimate of the reciprocal of the condition number of general matrix  $A$  is returned.

If  $n = 0$ , the subroutines return with  $rcond = 1.0$

Else if  $n \neq 0$  and  $anorm = 0.0$ , the subroutines return with  $rcond = 0.0$

Scope: **global**

Returned as: a long-precision real number;  $rcond \geq 0.0$ .

$info$  has the following meaning:

If  $info = 0$ , the computation completed normally.

Scope: **global**

Returned as: a fullword integer;  $info = 0$ .

## Notes and Coding Rules

1. In your C program, arguments  $rcond$  and  $info$  must be passed by reference.
2. This subroutine accepts lowercase letters for the  $norm$  argument.
3. The matrix and vector must have no common elements; otherwise, results are unpredictable.
4. The scalar data specified for input argument  $n$  must be the same for PDLANGE/PZLANGE, PDGETRF/PZGETRF, and PDGECON/PZGECON. In addition, the scalar data specified for input argument  $m$  in PDLANGE/PZLANGE and PDGETRF/PZGETRF **must be the same** as input argument  $n$  in PDLANGE/PZLANGE, PDGETRF/PZGETRF, and PDGECON/PZGECON.
5. The global submatrix for  $A$  input to PDLANGE/PZLANGE must be the same as the corresponding input arguments for PDGETRF/PZGETRF; and thus, the scalar data specified for  $ia$ ,  $ja$ , and the contents of  $desc_a$  must also be the same.
6. The global submatrix for  $A$  input to PDGECON/PZGECON must be the same as the corresponding output arguments for PDGETRF/PZGETRF; and thus, the scalar data specified for  $ia$ ,  $ja$ , and the contents of  $desc_a$  must also be the same.
7. The NUMROC utility subroutine can be used to determine the values of  $LOCp(M_)$  and  $LOCq(N_)$  used in the argument descriptions above. For details, see "Determining the Number of Rows and Columns in Your Local Arrays" on page 28 and "NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process" on page 868.
8. For suggested block sizes, see "Coding Tips for Optimizing Parallel Performance" on page 77.
9. On both input and output, matrix  $A$  conforms to ScaLAPACK format.
10. The global general matrix  $A$  must be distributed using a square block-cyclic distribution; that is,  $MB\_A = NB\_A$ .
11. The global general matrix  $A$  must be aligned on a block row boundary; that is,  $ia-1$  must be a multiple of  $MB\_A$ .
12. The block row offset of  $A$  must be equal to the block column offset of  $A$ ; that is,  $\text{mod}(ia-1, MB\_A) = \text{mod}(ja-1, NB\_A)$ .



## Error Conditions

### Computational Errors

None.

### Resource Errors

Unable to allocate work space.

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. DTYPE\_A is invalid.

#### Stage 2:

1. CTXT\_A is invalid.

#### Stage 3:

1. This subroutine was called from outside the process grid.

#### Stage 4:

1.  $norm \neq 'O', '1', \text{ or } 'T'$
2.  $n < 0$
3.  $M\_A < 0$  and  $n = 0$ ;  $M\_A < 1$  otherwise
4.  $N\_A < 0$  and  $n = 0$ ;  $N\_A < 1$  otherwise
5.  $anorm < 0$
6.  $ia < 1$
7.  $ja < 1$
8.  $MB\_A < 1$
9.  $NB\_A < 1$
10.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
11.  $CSRC\_A < 0$  or  $CSRC\_A \geq q$

#### Stage 5: If $n \neq 0$ :

1.  $ia > M\_A$
2.  $ja > N\_A$
3.  $ia+n-1 > M\_A$
4.  $ja+n-1 > N\_A$

In all cases:

5.  $MB\_A \neq NB\_A$
6.  $\text{mod}(ia-1, MB\_A) \neq \text{mod}(ja-1, NB\_A)$
7.  $\text{mod}(ia-1, MB\_A) \neq 0$

#### Stage 6:

1.  $LLD\_A < \max(1, \text{LOCp}(M\_A))$
2.  $lwork \neq 0$ ,  $lwork \neq -1$ , and  $lwork < (\text{minimum value})$ . (For the minimum value, see the *lwork* argument description.)
3.  $liwork \neq 0$ ,  $liwork \neq -1$ , and  $liwork < (\text{minimum value})$ . (For the minimum value, see the *liwork* argument description.)
4.  $lrwork \neq 0$ ,  $lrwork \neq -1$ , and  $lrwork < (\text{minimum value})$ . (For the minimum value, see the *lrwork* argument description.)

#### Stage 7:

Each of the following global input arguments are checked to determine whether its value differs from the value specified on process  $P_{00}$ :

1. *norm* differs.

## PDGECON and PZGECON

2.  $n$  differs.
3.  $ia$  differs.
4.  $ja$  differs.
5. DTYPE\_A differs.
6. M\_A differs.
7. N\_A differs.
8. MB\_A differs.
9. NB\_A differs.
10. RSRC\_A differs.
11. CSRC\_A differs.
12.  $anorm$  differs.

Also:

13.  $lwork = -1$  on a subset of processes.
14.  $liwork = -1$  on a subset of processes.
15.  $lrwork = -1$  on a subset of processes.

## Examples

### Example 1

This example estimates the reciprocal of the condition number of real general matrix  $A$ . The input matrix  $A$  to PDLANGE and PDGETRF is the same as input matrix  $A$  in the PDGETRF “Example 1” on page 374.

#### Notes:

1. Because WORK is used by both PDLANGE and PDGECON, we do not dynamically allocate the WORK area.
2. Because  $liwork = 0$ , PDGECON dynamically allocates the IWORK area used by this subroutine.

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      NORM, N    N    A    IA    JA    DESC_A    WORK
      |   |   |   |   |   |   |
ANORM = PDLANGE( 'O', 9 , 9,  A , 1 , 1, DESC_A, WORK )

      N    N    A    IA    JA    DESC_A    IPIV    INFO
      |   |   |   |   |   |   |
CALL PDGETRF( 9 , 9 , A , 1 , 1, DESC_A, IPIV, INFO )

      NORM  N    A    IA    JA    DESC_A    ANORM    RCOND    WORK    LWORK    IWORK    LIWORK    INFO
      |   |   |   |   |   |   |   |   |   |   |   |   |
CALL PDGECON( 'O', 9 , A , 1 , 1, DESC_A, ANORM, RCOND, WORK, 100 , IWORK , 0 , INFO )
```

	Desc_A
DTYPE_	1
CTXT_	<i>icontxt</i> <sup>1</sup>
M_	9
N_	9

	Desc_A
MB_	3
NB_	3
RSRC_	1
CSRC_	0
LLD_	See below <sup>2</sup>

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:  

$$\text{LLD\_A} = \text{MAX}(1, \text{NUMROC}(\text{M\_A}, \text{MB\_A}, \text{MYROW}, \text{RSRC\_A}, \text{NPROW}))$$
In this example, LLD\_A = 3 on P<sub>00</sub> and P<sub>01</sub>, and LLD\_A = 6 on P<sub>10</sub> and P<sub>11</sub>.

**Output:**

The value of *rcond* = 0.6173D-02 on all processes.

The value of *info* is 0 on all processes.

**Example 2**

This example estimates the reciprocal of the condition number of complex general matrix *A*. The input matrix *A* to PZLANGE and PZGETRF is the same as input matrix *A* in the PZGETRF “Example 2” on page 377.

**Notes:**

1. Because RWORK is used by both PZLANGE and PZGECON, we do not dynamically allocate the RWORK area.
2. Because *lwork* = 0, PZGECON dynamically allocates the WORK area used by this subroutine.

**Call Statements and Input:**

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      NORM, N   N   A   IA   JA   DESC_A   WORK
      |   |   |   |   |   |   |
ANORM = PZLANGE( 'O', 9 , 9,  A , 1 , 1, DESC_A, RWORK )

      N   N   A   IA   JA   DESC_A   IPIV   INFO
      |   |   |   |   |   |   |
CALL PZGETRF( 9 , 9 , A , 1 , 1, DESC_A, IPIV, INFO )

      NORM  N   A   IA   JA   DESC_A   ANORM  RCOND  WORK  LWORK  RWORK  LRWORK  INFO
      |   |   |   |   |   |   |   |   |   |   |   |   |
CALL PZGECON( 'O', 9,  A, 1, 1, DESC_A, ANORM, RCOND, WORK, 0,  RWORK, 100, INFO )
```

	Desc_A
DTYPE_	1

## PDGECON and PZGECON

	Desc_A
CTXT_	<i>icontxt</i> <sup>2</sup>
M_	9
N_	9
MB_	3
NB_	3
RSRC_	1
CSRC_	0
LLD_A	See below <sup>2</sup>

### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:  
$$\text{LLD\_A} = \text{MAX}(1, \text{NUMROC}(\text{M\_A}, \text{MB\_A}, \text{MYROW}, \text{RSRC\_A}, \text{NPROW}))$$
  
In this example, LLD\_A = 3 on P<sub>00</sub> and P<sub>01</sub>, and LLD\_A = 6 on P<sub>10</sub> and P<sub>11</sub>.

### Output:

The value of *rcond* = 0.3053D-01 on all processes.

The value of *info* is 0 on all processes.

## PDGEQRF and PZGEQRF — General Matrix QR Factorization

### Purpose

These subroutines compute the  $QR$  factorization of a general matrix  $A$ , where, in this description:

- $A$  represents the global general submatrix  $A_{ia:ia+m-1, ja:ja+n-1}$  to be factored.
- For PDGEQRF,  $Q$  is an orthogonal matrix.
- For PZGEQRF,  $Q$  is a unitary matrix.
- For  $m \geq n$ ,  $R$  is an upper triangular matrix.
- For  $m < n$ ,  $R$  is an upper trapezoidal matrix.

If  $m = 0$  or  $n = 0$ , no computation is performed and the subroutine returns after doing some parameter checking.

See references [24] and [39].

Table 75. Data Types

$A$ , $\tau$ , $work$	Subroutine
Long-precision real	PDGEQRF
Long-precision complex	PZGEQRF

### Syntax

<b>Fortran</b>	CALL PDGEQRF   PZGEQRF ( $m, n, a, ia, ja, desc\_a, tau, work, lwork, info$ )
<b>C and C++</b>	pdgeqrf   pzgeqrf ( $m, n, a, ia, ja, desc\_a, tau, work, lwork, info$ );

### On Entry

$m$  is the number of rows in submatrix  $A$  used in the computation.

Scope: **global**

Specified as: a fullword integer;  $m \geq 0$ .

$n$  is the number of columns in submatrix  $A$  used in the computation.

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

$a$  is the local part of the global general matrix  $A$ . This identifies the **first element** of the local array  $A$ . This subroutine computes the location of the first element of the local subarray used, based on  $ia, ja, desc\_a, p, q, myrow$ , and  $mycol$ ; therefore, the leading  $LOCp(ia+m-1)$  by  $LOCq(ja+n-1)$  part of the local array  $A$  must contain the local pieces of the leading  $ia+m-1$  by  $ja+n-1$  part of the global matrix.

Scope: **local**

Specified as: an  $LLD\_A$  by (at least)  $LOCq(N\_A)$  array, containing numbers of the data type indicated in Table 75. Details about the block-cyclic data distribution of global matrix  $A$  are stored in  $desc\_a$ .

$ia$  is the row index of the global matrix  $A$ , identifying the first row of the submatrix  $A$ .

## PDGEQRF and PZGEQRF

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq M\_A$  and  $ia+m-1 \leq M\_A$ .

*ja* is the column index of the global matrix *A*, identifying the first column of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$  and  $ja+n-1 \leq N\_A$ .

*desc\_a* is the array descriptor for global matrix *A*, described in the following table:

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $m = 0$ or $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	If $m = 0$ or $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_A < p$	Global
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < q$	Global
9	LLD_A	The leading dimension of the local array	$LLD\_A \geq \max(1, LOCp(M\_A))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

*tau* See On Return.

*work* has the following meaning:

If  $lwork = 0$ , *work* is ignored.

If  $lwork \neq 0$ , *work* is the work area used by this subroutine, where:

- If  $lwork \neq -1$ , its size is (at least) of length *lwork*.
- If  $lwork = -1$ , its size is (at least) of length 1.

Scope: **local**

Specified as: an area of storage containing numbers of data type indicated in Table 75 on page 411.

*lwork* is the number of elements in array WORK.

Scope:

- If  $lwork \geq 0$ , *lwork* is **local**
- If  $lwork = -1$ , *lwork* is **global**

Specified as: a fullword integer; where:

- If  $lwork = 0$ , PDGEQRF and PZGEQRF dynamically allocate the work area used by this subroutine. The work area is deallocated before control is returned to the calling program. This option is an extension to the ScaLAPACK standard.
- If  $lwork = -1$ , PDGEQRF and PZGEQRF perform a work area query and return the minimum size of  $work$  in  $work_1$ . No computation is performed and the subroutine returns after error checking is complete.
- Otherwise, it must have the following value:

$$lwork \geq nb (mp0 + nq0 + nb)$$

where:

- $mb = MB\_A$
- $nb = NB\_A$
- $iroff = \text{mod}(ia-1, mb)$
- $icoff = \text{mod}(ja-1, nb)$
- $iarow = \text{mod}(\text{RSRC\_A} + (ia-1)/mb, nrow)$
- $iacol = \text{mod}(\text{CSRC\_A} + (ja-1)/nb, ncol)$
- $mp0 = \text{NUMROC}(m+iroff, mb, myrow, iarow, nrow)$
- $nq0 = \text{NUMROC}(n+icoff, nb, mycol, iacol, ncol)$

*info* See On Return.

### On Return

*a* is the updated local part of the global general matrix  $A$ , containing the results of the computation.

The elements on and above the diagonal of  $A_{ia:ia+m-1, ja:ja+n-1}$  contain the  $\min(m, n) \times n$  upper trapezoidal matrix  $R$  ( $R$  is upper triangular if  $m \geq n$ ). The elements below the diagonal with  $\tau$  represent the matrix  $Q$  as a product of elementary reflectors.

Scope: **local**

Returned as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 75 on page 411. Details about the block-cyclic data distribution of global matrix  $A$  are stored in *desc\_a*.

*tau* is the updated local part of the global matrix  $\tau$ , where:  $\tau_{ja:ja+\min(m, n)-1}$  contains the scalar factors of the elementary reflectors.

This identifies the **first element** of the local array  $\tau$ . This subroutine computes the location of the first element of the local subarray used, based on *ja*, *desc\_a*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading 1 by LOCq( $ja+\min(m, n)-1$ ) part of the local array  $\tau$  must contain the local pieces of the leading 1 by  $ja+\min(m, n)-1$  part of the global matrix  $\tau$ .

A copy of the vector  $\tau$ , with a block size of NB\_A and global index *ja*, is returned to each row of the process grid. The process column over which the first column of  $\tau$  is distributed is CSRC\_A.

Scope: **local**

Returned as: a 1 by (at least) LOCq( $ja+\min(m, n)-1$ ) array, containing numbers of the data type indicated in Table 75 on page 411.

*work* is the work area used by this subroutine if  $lwork \neq 0$ , where:

If  $lwork \neq 0$  and  $lwork \neq -1$ , its size is (at least) of length *lwork*.

If  $lwork = -1$ , its size is (at least) of length 1.

Scope: **local**

Returned as: an area of storage, where:

If  $lwork \geq 1$  or  $lwork = -1$ , then  $work_1$  is set to the minimum  $lwork$  value and contains numbers of the data type indicated in Table 75 on page 411. Except for  $work_1$ , the contents of  $work$  are overwritten on return.

*info* indicates that a successful computation occurred.

Scope: **global**

Returned as: a fullword integer; *info* = 0.

## Notes and Coding Rules

1. In your C program, argument *info* must be passed by reference.
2. Matrix *A*,  $\tau$ , and *work* must have no common elements; otherwise, results are unpredictable.
3. The NUMROC utility subroutine can be used to determine the values of LOCp(M\_) and LOCq(N\_) used in the argument descriptions above. For details, see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28 and “NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process” on page 868.
4. There is no array descriptor for  $\tau$ .  $\tau$  is a row-distributed vector with block size NB\_A, local array of dimension 1 by LOCq( $ja + \min(m, n) - 1$ ), and global index *ja*. A copy of  $\tau$  exists on each row of the process grid, and the process column over which the first column of  $\tau$  is distributed is CSRC\_A.
5. For suggested block sizes, see “Coding Tips for Optimizing Parallel Performance” on page 77.
6. If  $lwork = -1$  on any process, it must equal  $-1$  on all processes. That is, if a subset of the processes specifies  $-1$  for the work area size, they must all specify  $-1$ .

## Function

These subroutines compute the *QR* factorization of a general matrix *A*.

$$A = QR$$

where:

- *A* represents the global general submatrix  $A_{ia:ia+m-1, ja:ja+n-1}$  to be factored.
- Matrix *Q* is represented as a product of elementary reflectors:

$$Q = H_{ja} H_{ja+1} \dots H_{ja+k-1}$$

where:

- $k = \min(m, n)$
- For each *i*, the following is true:

**For subroutine PDGEQRF:**

$$H_i = I - \tau v v^T$$

where:

- $\tau$  is a real scalar.
- *v* is a real vector with  $v_{1:i-1} = 0$ ,  $v_i = 1$ .

**For subroutine PZGEQRF:**

$$H_i = I - \tau v v^H$$

where:



- $\tau$  is a complex scalar.
- $v$  is a complex vector with  $v_{1:i-1} = (0,0)$ ,  $v_i = (1,0)$ .

**For both subroutines:**

$I$  is the identity matrix.

$v_{i+1:m}$  is stored on return in submatrix  $A_{ia+i: ia+m-1, ja+i-1}$ .

$\tau$  is stored on return in  $\tau_{ja+i-1}$ .

## Error Conditions

### Computational Errors

None

### Resource Errors

1.  $lwork = 0$  and unable to allocate work space

### Input-Argument and Miscellaneous Errors

**Stage 1:**

1.  $DTYPE\_A$  is invalid.

**Stage 2:**

1.  $CTXT\_A$  is invalid.

**Stage 3:**

1. This subroutine has been called from outside the process grid.

**Stage 4:**

1.  $m < 0$
2.  $n < 0$
3.  $M\_A < 0$  and ( $m = 0$  or  $n = 0$ );  $M\_A < 1$  otherwise
4.  $N\_A < 0$  and ( $m = 0$  or  $n = 0$ );  $N\_A < 1$  otherwise
5.  $ia < 1$
6.  $ja < 1$
7.  $MB\_A < 1$
8.  $NB\_A < 1$
9.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
10.  $CSRC\_A < 0$  or  $CSRC\_A \geq q$

**Stage 5:** If  $m \neq 0$  and  $n \neq 0$ :

1.  $ia > M\_A$
2.  $ja > N\_A$
3.  $ia+m-1 > M\_A$
4.  $ja+n-1 > N\_A$

**Stage 6:**

1.  $LLD\_A < \max(1, LOCp(M\_A))$
2.  $lwork \neq 0$ ,  $lwork \neq -1$ , and  $lwork < (nb (mp0 + nq0 + nb))$

where:

- $mb = MB\_A$
- $nb = NB\_A$
- $iroff = \text{mod}(ia-1, mb)$
- $icoff = \text{mod}(ja-1, nb)$
- $iarow = \text{mod}(RSRC\_A + (ia-1)/mb, nprow)$
- $iacol = \text{mod}(CSRC\_A + (ja-1)/nb, npcol)$
- $mp0 = \text{NUMROC}(m+iroff, mb, myrow, iarow, nprow)$

## PDGEQRF and PZGEQRF

- $nq0 = \text{NUMROC}(n+icoff, nb, mycol, iacol, npcol)$

### Stage 7:

Each of the following global input arguments are checked to determine whether its value differs from the value specified on process  $P_{00}$ :

1.  $m$  differs.
2.  $n$  differs.
3.  $ia$  differs.
4.  $ja$  differs.
5.  $\text{DTYPE\_A}$  differs.
6.  $\text{M\_A}$  differs.
7.  $\text{N\_A}$  differs.
8.  $\text{MB\_A}$  differs.
9.  $\text{NB\_A}$  differs.
10.  $\text{RSRC\_A}$  differs.
11.  $\text{CSRC\_A}$  differs.

Also:

12.  $lwork = -1$  on a subset of processes.

## Examples

### Example 1

This example shows the **QR** factorization of a real general matrix of size  $4 \times 3$ , using a  $2 \times 2$  process grid.

**Note:** Because  $lwork = 0$ , PDGEQRF dynamically allocates the work area used by this subroutine.

### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      M   N   A   IA   JA   DESC_A   TAU   WORK   LWORK   INFO
      |   |   |   |   |   |         |   |         |         |
CALL PDGEQRF( 4 , 3 , A , 1 , 1 , DESC_A , TAU , WORK , 0 , INFO)
```

	DESC_A
DTYPE_	1
CTXT_	<i>icontxt</i> <sup>1</sup>
M_	4
N_	3
MB_	1
NB_	1
RSRC_	0
CSRC_	0
LLD_	See below <sup>2</sup>

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:  

$$\text{LLD\_A} = \text{MAX}(1, \text{NUMROC}(\text{M\_A}, \text{MB\_A}, \text{MYROW}, \text{RSRC\_A}, \text{NPROW}))$$
In this example,  $\text{LLD\_A} = 2$  on all processes.

Global general matrix  $A$  of size  $4 \times 3$  with block sizes  $1 \times 1$ :

B,D	0	1	2
0	1.00	-2.00	-1.00
1	2.00	.00	1.00
2	2.00	-4.00	2.00
3	4.00	.00	.00

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$
3		

Local arrays for  $A$ :

p,q	0	1
0	1.00 -1.00 2.00 2.00	-2.00 -4.00
1	2.00 1.00 4.00 0.00	0.00 0.00

**Output:**

Global general matrix  $A$  of size  $4 \times 3$  with block sizes  $1 \times 1$ :

B,D	0	1	2
0	-5.00	2.00	-1.00
1	0.33	-4.00	1.00
2	0.33	-0.50	-2.00
3	0.67	0.50	0.00

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$
3		

Local arrays for  $A$ :

## PDGEQRF and PZGEQRF

p,q	0		1
0	-5.00	-1.00	2.00
	0.33	-2.00	-0.50
1	0.33	1.00	-4.00
	0.67	0.00	0.50

Global row vector  $\tau$  of length 3 with block size of 1:

B,D	0	1	2
0	1.20	1.33	2.00

**Note:** A copy of  $\tau$  is distributed across each row of the process grid.

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
	$P_{00}$	$P_{01}$
	$P_{10}$	$P_{11}$

Local arrays for  $\tau$ :

p,q	0		1
0	1.20	2.00	1.33
1	1.20	2.00	1.33

The value of *info* is 0 on all processes.

### Example 2

This example shows the **QR** factorization of a complex general matrix of size  $3 \times 4$ , using a  $2 \times 2$  process grid.

**Note:** Because *lwork* = 0, PZGEQRF dynamically allocates the work area used by this subroutine.

### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      M      N      A  IA  JA  DESC_A  TAU  WORK  LWORK  INFO
CALL PZGEQRF( 3 , 4 , A , 1 , 1 , DESC_A , TAU , WORK , 0 , INFO)
```

	DESC_A
DTYPE_	1
CTXT_	<i>icontxt</i> <sup>1</sup>
M_	3
N_	4

	DESC_A
MB_	1
NB_	1
RSRC_	0
CSRC_	0
LLD_	See below <sup>2</sup>

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:  

$$\text{LLD\_A} = \text{MAX}(1, \text{NUMROC}(\text{M\_A}, \text{MB\_A}, \text{MYROW}, \text{RSRC\_A}, \text{NPROW}))$$

$$= 2 \text{ on } P_{00} \text{ and } P_{01} \text{ and } 1 \text{ on } P_{10} \text{ and } P_{11}$$

Global general matrix  $A$  of size  $3 \times 4$  with block sizes  $1 \times 1$ :

$$\begin{array}{c}
 \begin{array}{ccccc}
 & 0 & 1 & 2 & 3 \\
 0 & \left[ \begin{array}{c|c|c|c} \hline \text{-----} & \text{-----} & \text{-----} & \text{-----} \\ \hline (1.00, 0.00) & (-2.00, 1.00) & (-3.00, -1.00) & (4.00, -3.00) \\ \hline \text{-----} & \text{-----} & \text{-----} & \text{-----} \\ \hline \end{array} \right. \\
 1 & \left[ \begin{array}{c|c|c|c} \hline \text{-----} & \text{-----} & \text{-----} & \text{-----} \\ \hline (1.00, -1.00) & (2.00, 2.00) & (-3.00, 0.00) & (-4.00, -2.00) \\ \hline \text{-----} & \text{-----} & \text{-----} & \text{-----} \\ \hline \end{array} \right. \\
 2 & \left[ \begin{array}{c|c|c|c} \hline \text{-----} & \text{-----} & \text{-----} & \text{-----} \\ \hline (1.00, -2.00) & (-2.00, 3.00) & (-3.00, 1.00) & (4.00, -1.00) \\ \hline \text{-----} & \text{-----} & \text{-----} & \text{-----} \\ \hline \end{array} \right. \\
 \end{array}
 \end{array}$$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

Local arrays for  $A$ :

p,q	0	1
0	$\begin{pmatrix} (1.00, 0.00) & (-3.00, -1.00) \\ (1.00, -2.00) & (-3.00, 1.00) \end{pmatrix}$	$\begin{pmatrix} (-2.00, 1.00) & (4.00, -3.00) \\ (-2.00, 3.00) & (4.00, -1.00) \end{pmatrix}$
1	$\begin{pmatrix} (1.00, -1.00) & (-3.00, 0.00) \end{pmatrix}$	$\begin{pmatrix} (2.00, 2.00) & (-4.00, -2.00) \end{pmatrix}$

**Output:** Global general matrix  $A$  of size  $3 \times 4$  with block sizes  $1 \times 1$ :

$$\begin{array}{c}
 \begin{array}{ccccc}
 & 0 & 1 & 2 & 3 \\
 0 & \left[ \begin{array}{c|c|c|c} \hline \text{-----} & \text{-----} & \text{-----} & \text{-----} \\ \hline (-2.83, 0.00) & (3.54, -1.41) & (3.89, 3.18) & (-2.83, 0.71) \\ \hline \text{-----} & \text{-----} & \text{-----} & \text{-----} \\ \hline \end{array} \right. \\
 1 & \left[ \begin{array}{c|c|c|c} \hline \text{-----} & \text{-----} & \text{-----} & \text{-----} \\ \hline (0.26, -0.26) & (-3.39, 0.00) & (0.37, -0.37) & (6.78, 0.74) \\ \hline \text{-----} & \text{-----} & \text{-----} & \text{-----} \\ \hline \end{array} \right. \\
 2 & \left[ \begin{array}{c|c|c|c} \hline \text{-----} & \text{-----} & \text{-----} & \text{-----} \\ \hline (0.26, -0.52) & (-0.29, -0.09) & (-1.87, 0.00) & (1.87, -1.87) \\ \hline \text{-----} & \text{-----} & \text{-----} & \text{-----} \\ \hline \end{array} \right. \\
 \end{array}
 \end{array}$$

The following is the  $2 \times 2$  process grid:

## PDGEQRF and PZGEQRF

B,D	0 2	1 3
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

Local arrays for  $A$ :

p,q	0	1
0	(-2.83, 0.00) ( 3.89, 3.18) ( 0.26,-0.52) (-1.87, 0.00)	( 3.54,-1.41) (-2.83, 0.71) (-0.29,-0.09) ( 1.87,-1.87)
1	( 0.26,-0.26) ( 0.37,-0.37)	(-3.39, 0.00) ( 6.78, 0.74)

Global row vector  $\tau$  of length 3 with block size of 1:

0	1	2
( 1.35, 0.00)	( 1.83,-0.02)	( 1.47,-0.88)

The following is the  $2 \times 2$  process grid.

**Note:** A copy of  $\tau$  is distributed across each row of the process grid.

B,D	0 2	1
	$P_{00}$	$P_{01}$
	$P_{10}$	$P_{11}$

Local arrays for  $\tau$ :

p,q	0	1
0	( 1.35, 0.00) ( 1.47,-0.88)	( 1.83,-0.02)
1	( 1.35, 0.00) ( 1.47,-0.88)	( 1.83,-0.02)

The value of *info* is 0 on all processes.

## PDGELS and PZGELS — General Matrix Least Squares Solution

### Purpose

PDGELS solves overdetermined or underdetermined real linear systems involving a real general matrix  $A$  or its transpose, using a  $QR$  or  $LQ$  factorization. It is assumed that  $A$  has full rank.

PZGELS solves overdetermined or underdetermined complex linear systems involving a complex general matrix  $A$  or its conjugate transpose, using a  $QR$  or  $LQ$  factorization. It is assumed that  $A$  has full rank.

The following options are provided:

- If  $transa = 'N'$  and  $m \geq n$ : find the least squares solution of an overdetermined system; that is, solve the least squares problem: minimize  $\|B - AX\|$
- If  $transa = 'N'$  and  $m < n$ : find the minimum norm solution of an underdetermined system; that is, the problem is:  $AX = B$
- For PDGELS:
  - If  $transa = 'T'$  and  $m \geq n$ : find the minimum norm solution of an underdetermined system; that is, the problem is  $A^T X = B$
  - If  $transa = 'T'$  and  $m < n$ : find the least squares solution of an overdetermined system; that is, solve the least squares problem: minimize  $\|B - A^T X\|$
- For PZGELS:
  - If  $transa = 'C'$  and  $m \geq n$ : find the minimum norm solution of an underdetermined system; that is, the problem is  $A^H X = B$
  - If  $transa = 'C'$  and  $m < n$ : find the least squares solution of an overdetermined system; that is, solve the least squares problem: minimize  $\|B - A^H X\|$

In the formulas above:

- $A$  represents the global general submatrix  $A_{ia:ia+m-1, ja:ja+n-1}$
- If  $transa = 'N'$ :
  - $B$  represents the global general submatrix  $B_{ib:ib+m-1, jb:jb+nrhs-1}$  containing the right-hand sides in its columns.
  - $X$  represents the global general submatrix  $B_{ib:ib+n-1, jb:jb+nrhs-1}$  containing the solution vectors in its columns.
- If  $transa \neq 'N'$ :
  - $B$  represents the global general submatrix  $B_{ib:ib+n-1, jb:jb+nrhs-1}$  containing the right-hand sides in its columns.
  - $X$  represents the global general submatrix  $B_{ib:ib+m-1, jb:jb+nrhs-1}$  containing the solution vectors in its columns.

**Note:** No data should be moved to form  $A^T$  or  $A^H$ ; that is, the matrix  $A$  should always be stored in its untransposed form.

If ( $m = 0$  and  $n = 0$ ) or  $nrhs = 0$ , then the subroutine returns after doing some parameter checking.

See references [13] and [39].

Table 76. Data Types

$A, B, work$	Subroutine
--------------	------------

Table 76. Data Types (continued)

Long-precision real	PDGELS
Long-precision complex	PZGELS

## Syntax

Fortran	CALL PDGELS PZGELS ( <i>transa</i> , <i>m</i> , <i>n</i> , <i>nrhs</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>b</i> , <i>ib</i> , <i>jb</i> , <i>desc_b</i> , <i>work</i> , <i>lwork</i> , <i>info</i> )
C and C++	pdgels pzgels ( <i>transa</i> , <i>m</i> , <i>n</i> , <i>nrhs</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>b</i> , <i>ib</i> , <i>jb</i> , <i>desc_b</i> , <i>work</i> , <i>lwork</i> , <i>info</i> );

### On Entry

*transa* indicates the form of matrix *A* used in the system of equations, where:

If *transa* = 'N', matrix *A* is used.

If *transa* = 'T', matrix  $A^T$  is used.

If *transa* = 'C', matrix  $A^H$  is used.

Scope: **global**

Specified as: a single character; *transa* = 'N', 'T', or 'C'.

*m* is the number of rows in submatrix *A* used in the computation.

Scope: **global**

Specified as: a fullword integer;  $m \geq 0$ .

*n* is the number of columns in submatrix *A* used in the computation.

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

*nrhs* is the number of right-hand sides; that is the number of columns in submatrices used in the computation.

Scope: **global**

Specified as: a fullword integer;  $nrhs \geq 0$ .

*a* is the local part of the global general matrix *A*. This identifies the **first element** of the local array *A*. This subroutine computes the location of the first element of the local subarray used, based on *ia*, *ja*, *desc\_a*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading LOCp(*ia+m-1*) by LOCq(*ja+n-1*) part of the local array *A* must contain the local pieces of the leading *ia+m-1* by *ja+n-1* part of the global matrix.

**Note:** No data should be moved to form  $A^T$  or  $A^H$ ; that is, the matrix *A* should always be stored in its untransposed form.

Scope: **local**

Specified as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 76 on page 421. Details about the block-cyclic data distribution of global matrix *A* are stored in *desc\_a*.

*ia* is the row index of the global matrix *A*, identifying the first row of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq M\_A$  and  $ia+m-1 \leq M\_A$ .



$ja$  is the column index of the global matrix  $A$ , identifying the first column of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$  and  $ja+n-1 \leq N\_A$ .

$desc\_a$  is the array descriptor for global matrix  $A$ , described in the following table:

$desc\_a$	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $m = 0$ or $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	If $m = 0$ or $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_A < p$	Global
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < q$	Global
9	LLD_A	The leading dimension of the local array	$LLD\_A \geq \max(1, LOCp(M\_A))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

$b$  is the local part of the global general matrix  $B$ , containing the right-hand sides of the system. This identifies the **first element** of the local array  $B$ . This subroutine computes the location of the first element of the local subarray used, based on  $ib$ ,  $jb$ ,  $desc\_b$ ,  $p$ ,  $q$ ,  $myrow$ , and  $mycol$ .

If  $transa = 'N'$ , the leading  $LOCp(ib+m-1)$  by  $LOCq(jb+nrhs-1)$  part of the local array  $B$  must contain the local pieces of the leading  $ib+m-1$  by  $jb+nrhs-1$  part of the global matrix; otherwise, the leading  $LOCp(ib+n-1)$  by  $LOCq(jb+nrhs-1)$  part of the local array  $B$  must contain the local pieces of the leading  $ib+n-1$  by  $jb+nrhs-1$  part of the global matrix.

Scope: **local**

Specified as: an  $LLD\_B$  by (at least)  $LOCq(N\_B)$  array, containing numbers of the data type indicated in Table 76 on page 421. Details about the square block-cyclic data distribution of global matrix  $B$  are stored in  $desc\_b$ .

$ib$  is the row index of the global matrix  $B$ , identifying the first row of the submatrix  $B$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ib \leq M\_B$  and  $ib + \max(m, n)-1 \leq M\_B$ .

## PDGELS and PZGELS

*jb* is the column index of the global matrix *B*, identifying the first column of the submatrix *B*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq jb \leq N\_B$  and  $jb+nrhs-1 \leq N\_B$ .

*desc\_b* is the array descriptor for global matrix *B*, described in the following table:

<i>desc_b</i>	Name	Description	Limits	Scope
1	DTYPE_B	Descriptor type	DTYPE_B=1	Global
2	CTXT_B	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_B	Number of rows in the global matrix	If ( $m = 0$ and $n = 0$ ) or ( $nrhs = 0$ ): $M\_B \geq 0$ Otherwise: $M\_B \geq 1$	Global
4	N_B	Number of columns in the global matrix	If ( $m = 0$ and $n = 0$ ) or ( $nrhs = 0$ ): $N\_B \geq 0$ Otherwise: $N\_B \geq 1$	Global
5	MB_B	Row block size	$MB\_B \geq 1$	Global
6	NB_B	Column block size	$NB\_B \geq 1$	Global
7	RSRC_B	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_B < p$	Global
8	CSRC_B	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_B < q$	Global
9	LLD_B	The leading dimension of the local array	$LLD\_B \geq \max(1, LOCp(M\_B))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

*work* has the following meaning:

If  $lwork = 0$ , *work* is ignored.

If  $lwork \neq 0$ , *work* is the work area used by this subroutine, where:

- If  $lwork \neq -1$ , its size is (at least) of length *lwork*.
- If  $lwork = -1$ , its size is (at least) of length 1.

Scope: **local**

Specified as: an area of storage containing numbers of data type indicated in Table 76 on page 421.

*lwork* is the number of elements in array WORK.

Scope:

- If  $lwork \geq 0$ , *lwork* is **local**
- If  $lwork = -1$ , *lwork* is **global**

Specified as: a fullword integer; where:

- If  $lwork = 0$ , PDGELS and PZGELS dynamically allocate the work area used by this subroutine. The work area is deallocated before control is returned to the calling program. This option is an extension to the ScaLAPACK standard.
- If  $lwork = -1$ , PDGELS and PZGELS dynamically perform a work area query and returns the minimum size of  $work$  in  $work_1$ . No computation is performed and the subroutine returns after error checking is complete.
- Otherwise, it must have the following value:

$$lwork \geq ltau + \max(lwf, lws)$$

where:

- If  $m \geq n$ , then:
  - $ltau = \text{NUMROC}(ja + \min(m, n) - 1, \text{NB\_A}, \text{mycol}, \text{CSRC\_A}, \text{npcol})$
  - $lwf = \text{NB\_A} (\text{mpa0} + \text{nqa0} + \text{NB\_A})$
  - $lws = \max((\text{NB\_A} (\text{NB\_A} - 1)) / 2, (\text{nrhsqb0} + \text{mpb0}) \text{NB\_A} + (\text{NB\_A})(\text{NB\_A}))$
- If  $m < n$ , then:
  - $ltau = \text{NUMROC}(ia + \min(m, n) - 1, \text{MB\_A}, \text{myrow}, \text{RSRC\_A}, \text{nproW})$
  - $lwf = \text{MB\_A} (\text{mpa0} + \text{nqa0} + \text{MB\_A})$
  - $lws = \max((\text{MB\_A} (\text{MB\_A} - 1)) / 2, (\text{npb0} + \max(\text{nqa0} + \text{NUMROC}(\text{NUMROC}(n + \text{irowfb}, \text{MB\_A}, 0, 0, \text{nproW}), \text{MB\_A}, 0, 0, \text{lcmp}), \text{nrhsqb0})) \text{MB\_A} + (\text{MB\_A})(\text{MB\_A}))$

where:

- $lcm = \text{ilcm}(\text{nproW}, \text{npcol})$
- $lcmp = lcm / \text{nproW}$
- $\text{irowfb} = \text{mod}(ia - 1, \text{MB\_A})$
- $\text{icoffa} = \text{mod}(ja - 1, \text{NB\_A})$
- $\text{irow} = \text{mod}(\text{RSRC\_A} + (\text{ia} - 1) / \text{MB\_A}, \text{nproW})$
- $\text{icol} = \text{mod}(\text{CSRC\_A} + (\text{ja} - 1) / \text{NB\_A}, \text{npcol})$
- $\text{mpa0} = \text{NUMROC}(m + \text{irowfb}, \text{MB\_A}, \text{myrow}, \text{irow}, \text{nproW})$
- $\text{nqa0} = \text{NUMROC}(n + \text{icoffa}, \text{NB\_A}, \text{mycol}, \text{icol}, \text{npcol})$
- $\text{irowfb} = \text{mod}(ib - 1, \text{MB\_B})$
- $\text{icoffb} = \text{mod}(jb - 1, \text{NB\_B})$
- $\text{ibrow} = \text{mod}(\text{RSRC\_B} + (\text{ib} - 1) / \text{MB\_B}, \text{nproW})$
- $\text{ibcol} = \text{mod}(\text{CSRC\_B} + (\text{jb} - 1) / \text{NB\_B}, \text{npcol})$
- $\text{mpb0} = \text{NUMROC}(m + \text{irowfb}, \text{MB\_B}, \text{myrow}, \text{ibrow}, \text{nproW})$
- $\text{npb0} = \text{NUMROC}(n + \text{icoffb}, \text{NB\_B}, \text{mycol}, \text{ibcol}, \text{npcol})$
- $\text{nrhsqb0} = \text{NUMROC}(\text{nrhs} + \text{icoffb}, \text{NB\_B}, \text{mycol}, \text{ibcol}, \text{npcol})$

*info* See On Return.

## On Return

- a* is the updated local part of the global general matrix *A*. Matrix *A* is overwritten; the original input is not preserved.

Scope: **local**

Returned as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 76 on page 421. Details about the block-cyclic data distribution of global matrix *A* are stored in *desc\_a*.

- b* is the updated local part of the global general matrix *B*, overwritten by the solution vectors stored columnwise.
- If *transa* = 'N' and  $m \geq n$ , rows *ib:ib+n-1* contain the least squares solution vectors. The residual sum of squares for each column is given by the sum of squares of elements *ib+n:ib+m-1* in that column.

- If *transa* = 'N' and  $m < n$ , rows  $ib:ib+n-1$  contain the minimum norm solution vectors.
- If *transa*  $\neq$  'N' and  $m \geq n$ , rows  $ib:ib+m-1$  contain the minimum norm solution vectors.
- If *transa*  $\neq$  'N' and  $m < n$ , rows  $ib:ib+m-1$  contain the least squares solution vectors. The residual sum of squares for each column is given by the sum of squares of elements  $ib+m:ib+n-1$  in that column.

Scope: **local**

Returned as: an LLD\_B by (at least) LOCq(N\_B) array, containing numbers of the data type indicated in Table 76 on page 421. Details about the block-cyclic data distribution of global matrix *B* are stored in *desc\_b*.

*work* is the work area used by this subroutine if *lwork*  $\neq$  0.

Scope: **local**

Returned as: an area of storage, where:

If *lwork*  $\geq$  1 or *lwork* = -1, then *work*<sub>1</sub> is set to the minimum *lwork* value and contains numbers of the data type indicated in Table 76 on page 421. Except for *work*<sub>1</sub>, the contents of *work* are overwritten on return.

*info* indicates that a successful computation occurred.

Scope: **global**

Returned as: a fullword integer; *info* = 0.

## Notes and Coding Rules

1. This subroutine accepts lowercase letters for the *transa* argument.
2. In your C program, argument *info* must be passed by reference.
3. Matrices *A*, *B*, and *work* must have no common elements; otherwise, results are unpredictable.
4. The NUMROC utility subroutine can be used to determine the values of LOCp(M\_) and LOCq(N\_) used in the argument descriptions above. For details, see "Determining the Number of Rows and Columns in Your Local Arrays" on page 28 and "NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process" on page 868.
5. For suggested block sizes, see "Coding Tips for Optimizing Parallel Performance" on page 77.
6. The following values must be equal: CTXT\_A = CTXT\_B
7. If  $m \geq n$ :
  - The following block sizes must be equal: MB\_A = MB\_B
  - The row block offset of *A* must be equal to the row block offset of *B*; that is,  $\text{mod}(ia-1, MB_A) = \text{mod}(ib-1, MB_B)$
  - In the process grid, the process row containing the first row of the submatrix *A* must also contain the first row of the submatrix *B*; that is, *iarow* = *ibrow*, where:
    - *iarow* =  $\text{mod}(\text{RSRC\_A} + (ia-1)/MB\_A, p)$
    - *ibrow* =  $\text{mod}(\text{RSRC\_B} + (ib-1)/MB\_B, p)$
8. If  $m < n$ :
  - The following block sizes must be equal: NB\_A = MB\_B
  - The column block offset of *A* must be equal to the row block offset of *B*; that is,  $\text{mod}(ja-1, NB\_A) = \text{mod}(ib-1, MB\_B)$

9. If  $m < n$  and  $m \neq 0$ ,  $n \neq 0$ , and  $nrhs \neq 0$ :
  - In the process grid, the process row containing the first row of the submatrix  $A$  must also contain the first row of the submatrix  $B$ ; that is,  $iarow = ibrow$ , where:
    - $iarow = \text{mod}(\text{RSRC\_A} + (ia-1)/\text{MB\_A}, p)$
    - $ibrow = \text{mod}(\text{RSRC\_B} + (ib-1)/\text{MB\_B}, p)$
10. If  $m \neq 0$ ,  $n \neq 0$ , and  $nrhs \neq 0$  and if  $A_{ia:ia+\min(m, n)-1, ja:ja+\min(m, n)-1}$  is **not** contained in a single block, that is, either of the following is true:
  - $\min(m, n) + \text{mod}(ia-1, \text{MB\_A}) > \text{MB\_A}$
  - $\min(m, n) + \text{mod}(ja-1, \text{NB\_A}) > \text{NB\_A}$
 then:
  - The global matrix  $A$  must be distributed using a square block-cyclic distribution; that is,  $\text{MB\_A} = \text{NB\_A}$ .
  - The submatrix  $A$  must be aligned on a block boundary, that is,
    - $ia-1$  must be a multiple of  $\text{MB\_A}$
    - $ja-1$  must be a multiple of  $\text{NB\_A}$
  - The submatrix  $B$  must be aligned on a block boundary, that is,
    - $ib-1$  must be a multiple of  $\text{MB\_B}$
11. If  $lwork = -1$  on any process, it must equal  $-1$  on all processes. That is, if a subset of the processes specifies  $-1$  for the work area size, they must all specify  $-1$ .

## Function

PDGELS solves overdetermined or underdetermined real linear systems involving a real general rectangular matrix  $A$ , or its transpose, using a  $QR$  or  $LQ$  factorization. It is assumed that  $A$  has full rank.

PZGELS solves overdetermined or underdetermined complex linear systems involving a complex general matrix  $A$  or its conjugate transpose, using a  $QR$  or  $LQ$  factorization. It is assumed that  $A$  has full rank.

The following options are provided:

- If  $transa = 'N'$  and  $m \geq n$ : find the least squares solution of an overdetermined system; that is, solve the least squares problem: minimize  $\|B - AX\|$
- If  $transa = 'N'$  and  $m < n$ : find the minimum norm solution of an underdetermined system; that is, the problem is:  $AX = B$
- For PDGELS:
  - If  $transa = 'T'$  and  $m \geq n$ : find the minimum norm solution of an underdetermined system; that is, the problem is  $A^T X = B$
  - If  $transa = 'T'$  and  $m < n$ : find the least squares solution of an overdetermined system; that is, solve the least squares problem: minimize  $\|B - A^T X\|$
- For PZGELS:
  - If  $transa = 'C'$  and  $m \geq n$ : find the minimum norm solution of an underdetermined system; that is, the problem is  $A^H X = B$
  - If  $transa = 'C'$  and  $m < n$ : find the least squares solution of an overdetermined system; that is, solve the least squares problem: minimize  $\|B - A^H X\|$

In the formulas above:

- $A$  represents the global general submatrix  $A_{ia:ia+m-1, ja:ja+n-1}$
- If  $transa = 'N'$ :

- $B$  represents the global general submatrix  $B_{ib:ib+m-1, jb:jb+nrhs-1}$  containing the right-hand sides in its columns.
- $X$  represents the global general submatrix  $B_{ib:ib+n-1, jb:jb+nrhs-1}$  containing the solution vectors in its columns.
- If  $transa \neq 'N'$ :
  - $B$  represents the global general submatrix  $B_{ib:ib+n-1, jb:jb+nrhs-1}$  containing the right-hand sides in its columns.
  - $X$  represents the global general submatrix  $B_{ib:ib+m-1, jb:jb+nrhs-1}$  containing the solution vectors in its columns.

**Note:** No data should be moved to form  $A^T$  or  $A^H$ ; that is, the matrix  $A$  should always be stored in its untransposed form.

If ( $m = 0$  and  $n = 0$ ) or  $nrhs = 0$ , then the subroutine returns after doing some parameter checking.

See references [13] and [39].

## Error Conditions

### Computational Errors

None

### Resource Errors

1.  $lwork = 0$  and unable to allocate work space

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1.  $DTYPE\_A$  is invalid.
2.  $DTYPE\_B$  is invalid.

#### Stage 2:

1.  $CTXT\_A$  is invalid.

#### Stage 3:

1. This subroutine has been called from outside the process grid.

#### Stage 4:

1.  $transa \neq$ 
  - $'N'$  or  $'T'$  for PDGELS
  - $'N'$  or  $'C'$  for PZGELS
2.  $m < 0$
3.  $n < 0$
4.  $nrhs < 0$
5.  $M\_A < 0$  and ( $m = 0$  or  $n = 0$ );  $M\_A < 1$  otherwise
6.  $N\_A < 0$  and ( $m = 0$  or  $n = 0$ );  $N\_A < 1$  otherwise
7.  $MB\_A < 1$
8.  $NB\_A < 1$
9.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
10.  $CSRC\_A < 0$  or  $CSRC\_A \geq q$
11.  $ia < 1$
12.  $ja < 1$
13.  $M\_B < 0$  and ( $(m = 0$  and  $n = 0)$  or  $nrhs = 0$ );  $M\_B < 1$  otherwise
14.  $N\_B < 0$  and ( $(m = 0$  and  $n = 0)$  or  $nrhs = 0$ );  $N\_B < 1$  otherwise
15.  $MB\_B < 1$
16.  $NB\_B < 1$

17.  $\text{RSRC\_B} < 0$  or  $\text{RSRC\_B} \geq p$
18.  $\text{CSRC\_B} < 0$  or  $\text{CSRC\_B} \geq q$
19.  $ib < 1$
20.  $jb < 1$
21.  $\text{CTXT\_A} \neq \text{CTXT\_B}$

**Stage 5:** If  $m \neq 0$ ,  $n \neq 0$ , and  $nrhs \neq 0$  and if  $A_{ia:ia+\min(m, n)-1, ja:ja+\min(m, n)-1}$  is **not** contained in a single block, that is, either of the following is true:

- $\min(m, n) + \text{mod}(ia-1, \text{MB\_A}) > \text{MB\_A}$
  - $\min(m, n) + \text{mod}(ja-1, \text{NB\_A}) > \text{NB\_A}$
1.  $\text{MB\_A} \neq \text{NB\_A}$
  2.  $\text{MB\_B} \neq \text{NB\_A}$

If  $m \neq 0$  and  $n \neq 0$ :

1.  $ia > \text{M\_A}$
2.  $ja > \text{M\_A}$
3.  $ia+m-1 > \text{M\_A}$
4.  $ja+n-1 > \text{N\_A}$

If  $(m \neq 0$  or  $n \neq 0)$  and  $nrhs \neq 0$ :

1.  $ib > \text{M\_B}$
2.  $jb > \text{M\_B}$
3.  $ib+m-1 > \text{M\_B}$  and  $m \geq n$ ;  $ib+n-1 > \text{M\_B}$  and  $m < n$
4.  $jb+nrhs-1 > \text{N\_B}$

If  $A_{ia:ia+\min(m, n)-1, ja:ja+\min(m, n)-1}$  is **not** contained in a single block and  $(m \neq 0, n \neq 0, \text{ and } nrhs \neq 0)$ :

1.  $\text{mod}(ia-1, \text{MB\_A}) \neq 0$
2.  $\text{mod}(ja-1, \text{NB\_A}) \neq 0$
3.  $\text{mod}(ib-1, \text{MB\_B}) \neq 0$

If  $m \geq n$ :

1.  $\text{MB\_A} \neq \text{MB\_B}$
2.  $\text{mod}(ia-1, \text{MB\_A}) \neq \text{mod}(ib-1, \text{MB\_B})$

If  $m < n$ :

1.  $\text{NB\_A} \neq \text{MB\_B}$
2.  $\text{mod}(ja-1, \text{NB\_A}) \neq \text{mod}(ib-1, \text{MB\_B})$

**Stage 6:**

1.  $\text{LLD\_A} < \max(1, \text{LOCp}(\text{M\_A}))$
2.  $\text{LLD\_B} < \max(1, \text{LOCp}(\text{M\_B}))$

If  $m \geq n$ :

1. In the process grid, the process row containing the first row of the submatrix  $A$  does not contain the first row of the submatrix  $B$ ; that is,  $iarow \neq ibrow$ , where:
  - $iarow = \text{mod}(\text{RSRC\_A} + (ia-1)/\text{MB\_A}, p)$
  - $ibrow = \text{mod}(\text{RSRC\_B} + (ib-1)/\text{MB\_B}, p)$

If  $m < n$  and  $(m \neq 0, n \neq 0, \text{ and } nrhs \neq 0)$ :

1. In the process grid, the process row containing the first row of the submatrix  $A$  does not contain the first row of the submatrix  $B$ ; that is,  $iarow \neq ibrow$ , where:
  - $iarow = \text{mod}(\text{RSRC\_A} + (ia-1)/\text{MB\_A}, p)$
  - $ibrow = \text{mod}(\text{RSRC\_B} + (ib-1)/\text{MB\_B}, p)$

In all cases:

## PDGELS and PZGELS

1.  $lwork \neq 0$ ,  $lwork \neq -1$ , and  $lwork < (\text{minimum value})$  (For minimum value, see *lwork* parameter description.)

### Stage 7:

Each of the following global input arguments are checked to determine whether its value differs from the value specified on process  $P_{00}$ :

1. *transa* differs.
2. *m* differs.
3. *n* differs.
4. *nrhs* differs.
5. *ia* differs.
6. *ja* differs.
7. DTYPE\_A differs.
8. M\_A differs.
9. N\_A differs.
10. MB\_A differs.
11. NB\_A differs.
12. RSRC\_A differs.
13. CSRC\_A differs.
14. *ib* differs.
15. *jb* differs.
16. DTYPE\_B differs.
17. M\_B differs.
18. N\_B differs.
19. MB\_B differs.
20. NB\_B differs.
21. RSRC\_B differs.
22. CSRC\_B differs.

Also:

23. *lwork* = -1 on a subset of processes.

## Examples

### Example 1

This example illustrates the least squares solution of an overdetermined real general system of size  $4 \times 3$  with 5 right hand sides, using a  $2 \times 2$  process grid.

**Note:** Because *lwork* = 0, PDGELS dynamically allocates the work area used by this subroutine.

### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      TRANSA  M    N  NRHS  A  IA  JA  DESC_A  B  IB  JB  DESC_B  WORK  LWORK  INFO
      |      |    |    |    |  |  |  |      |  |  |  |      |    |    |
CALL PDGELS ( 'N' , 4 , 3 , 5 , A , 1 , 1 , DESC_A , B , 1 , 1 , DESC_B , WORK , 0 , INFO )
```

	DESC_A	DESC_B
DTYPE_	1	1



	DESC_A	DESC_B
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	4	4
N_	3	5
MB_	1	1
NB_	1	2
RSRC_	0	0
CSRC_	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:  

$$\text{LLD\_A} = \text{MAX}(1, \text{NUMROC}(\text{M\_A}, \text{MB\_A}, \text{MYROW}, \text{RSRC\_A}, \text{NPROW}))$$

$$\text{LLD\_B} = \text{MAX}(1, \text{NUMROC}(\text{M\_B}, \text{MB\_B}, \text{MYROW}, \text{RSRC\_B}, \text{NPROW}))$$
 In this example, LLD\_A and LLD\_B = 2 on all processes.

Global general matrix *A* of size 4 by 3, with block sizes  $1 \times 1$ :

B,D	0	1	2
0	1.00	-2.00	-1.00
1	2.00	.00	1.00
2	2.00	-4.00	2.00
3	4.00	.00	.00

Global general matrix *B* of size 4 by 5, with block sizes  $1 \times 2$ :

B,D	0	1	2
0	-1.00 -2.00	-7.00 0.00	-5.00
1	1.00 3.00	4.00 3.00	5.00
2	1.00 0.00	4.00 2.00	2.00
3	-2.00 4.00	4.00 0.00	4.00

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	P <sub>00</sub>	P <sub>01</sub>
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for *A*:

p,q	0	1
0	1.00 -1.00	-2.00

	2.00	2.00	-4.00
1	2.00	1.00	0.00
	4.00	0.00	0.00

Local arrays for  $B$ :

p,q	0	1
0	-1.00 -2.00 -5.00 1.00 0.00 2.00	-7.00 0.00 4.00 2.00
1	1.00 3.00 5.00 -2.00 4.00 4.00	4.00 3.00 4.00 0.00

**Output:**

Global general matrix  $B$  of size 4 by 5, with block sizes  $1 \times 2$ :

B,D	0	1	2
0	-0.40 1.00	0.80 0.20	1.00
1	0.00 1.00	1.50 0.00	1.50
2	1.00 1.00	4.00 1.00	3.00
3	-1.00 0.00	2.00 -2.00	0.00

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$
3		

Local arrays for  $B$ :

p,q	0	1
0	-0.40 1.00 1.00 1.00 1.00 3.00	0.80 0.20 4.00 1.00
1	0.00 1.00 1.50 -1.00 0.00 0.00	1.50 0.00 2.00 -2.00

The value of *info* is 0 on all processes.

## Example 2

This example illustrates the least squares solution of an underdetermined complex general system of size  $3 \times 4$  with 3 right hand sides, using a  $2 \times 2$  process grid.

**Note:** Because  $lwork = 0$ , PZGELS dynamically allocates the work area used by this subroutine.

## Call Statements and Input:

```

ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      TRANSA  M    N  NRHS  A  IA  JA  DESC_A  B  IB  JB  DESC_B  WORK  LWORK  INFO
      |      |    |    |    |  |  |  |      |  |  |  |      |    |    |
CALL PZGELS ( 'N' , 3 , 4 , 3 , A , 1 , 1 , DESC_A , B , 1 , 1 , DESC_B , WORK , 0 , INFO )

```

	DESC_A	DESC_B
DTYPE_	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	3	4
N_	4	3
MB_	1	1
NB_	1	1
RSRC_	0	0
CSRC_	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>

## Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:  

$$\text{LLD\_A} = \text{MAX}(1, \text{NUMROC}(\text{M\_A}, \text{MB\_A}, \text{MYROW}, \text{RSRC\_A}, \text{NPROW}))$$

$$= 2 \text{ on } P_{00} \text{ and } P_{01} \text{ and } 1 \text{ on } P_{10} \text{ and } P_{11}$$

$$\text{LLD\_B} = \text{MAX}(1, \text{NUMROC}(\text{M\_B}, \text{MB\_B}, \text{MYROW}, \text{RSRC\_B}, \text{NPROW}))$$

$$= 2 \text{ on all processes}$$

Global general matrix  $A$  of size 3 by 4, with block sizes  $1 \times 1$ :

B,D	0	1	2	3
0	( 1.00, 0.00)	(-2.00, 1.00)	(-3.00,-1.00)	( 4.00,-3.00)
1	( 1.00,-1.00)	( 2.00, 2.00)	(-3.00, 0.00)	(-4.00,-2.00)
2	( 1.00,-2.00)	(-2.00, 3.00)	(-3.00, 1.00)	( 4.00,-1.00)

Global general matrix  $B$  of size 4 by 3, with block sizes  $1 \times 1$ :

B,D	0	1	2
0	( 1.00, .00)	( .00, 1.00)	( 1.00, 1.00)
1	(-1.00,1.00)	(1.00,-1.00)	( .00, .00)
2	2.00,1.00	(1.00, 2.00)	(-1.00,-1.00)
3	( . , . )	( . , . )	( . , . )

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>
3		

Local arrays for *A*:

p,q	0	1
0	(1.00, 0.00) (-3.00,-1.00) (1.00,-2.00) (-3.00, 1.00)	(-2.00, 1.00) ( 4.00,-3.00) (-2.00, 3.00) ( 4.00,-1.00)
1	(1.00,-1.00) (-3.00, 0.00)	( 2.00, 2.00) (-4.00,-2.00)

Local arrays for *B*:

p,q	0	1
0	( 1.00, .00) ( 1.00, 1.00) ( 2.00, 1.00) (-1.00,-1.00)	( 1.00,-1.00) ( 1.00, 2.00)
1	(-1.00, 1.00) ( .00, .00) ( . , . ) ( . , . )	( 1.00,-1.00) ( . , . )

**Output:**

Global general matrix *B* of size 4 by 3, with block sizes 1 × 1:

B,D	0	1	2
0	( -.16, .15)	( -.08, .18)	( .16, -.31)
1	( .11, .02)	( .21, -.50)	( -.38, .65)
2	( -.13, -.32)	( .16, .12)	( -.27, -.28)
3	( .37, -.05)	( .04, .06)	( -.19, .32)

The following is the 2 × 2 process grid:

B,D	0 2	1
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>
3		

Local arrays for *B*:

p,q	0	1
0	( -.16, .15) ( .16, -.31) ( -.13, -.32) ( -.27, -.28)	( -.08, .18) ( .16, .12)
1	( .11, .02) ( -.38, .65) ( .37, -.05) ( -.19, .32)	( .21, -.50) ( .04, .06)

The value of *info* is 0 on all processes.

## PDPOSV and PZPOSV — Positive Definite Real Symmetric or Complex Hermitian Matrix Factorization and Solve

### Purpose

These subroutines solve the following systems of equations for multiple right-hand sides:

- $AX = B$

where, in the formula above:

- $A$  represents the global positive definite real symmetric or complex Hermitian submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$ .
- $B$  represents the global general submatrix  $B_{ib:ib+n-1, jb:jb+nrhs-1}$  containing the right-hand sides in its columns.
- $X$  represents the global general submatrix  $B_{ib:ib+n-1, jb:jb+nrhs-1}$  containing the solution vectors in its columns.

If  $n$  is 0, no computation is performed and the subroutine returns after doing some parameter checking. If  $n > 0$  and  $nrhs$  is 0, no solutions are computed and the subroutine returns after factoring the matrix.

See references [17], [19], [23], [38], and [39].

Table 77. Data Types

$A, B$	Subroutine
Long-precision real	PDPOSV
Long-precision complex	PZPOSV

### Syntax

<b>Fortran</b>	CALL PDPOSV   PZPOSV ( <i>uplo, n, nrhs, a, ia, ja, desc_a, b, ib, jb, desc_b, info</i> )
<b>C and C++</b>	pdposv   pzposv ( <i>uplo, n, nrhs, a, ia, ja, desc_a, b, ib, jb, desc_b, info</i> );

### On Entry

*uplo* indicates whether the upper or lower triangular part of the global real symmetric or complex Hermitian submatrix  $A$  is referenced, where:

If *uplo* = 'U', the upper triangular part is referenced.

If *uplo* = 'L', the lower triangular part is referenced.

Scope: **global**

Specified as: a single character; *uplo* = 'U' or 'L'.

*n* is the order of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

*nrhs* is the number of right-hand sides— that is, the number of columns in submatrix  $B$  used in the computation.

Scope: **global**

Specified as: a fullword integer;  $nrhs \geq 0$ .

*a* is the local part of the global real symmetric or complex Hermitian matrix *A*, used in the system of equations. This identifies the **first element** of the local array *A*. This subroutine computes the location of the first element of the local subarray used, based on *ia*, *ja*, *desc\_a*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading LOCp(*ia*+*n*-1) by LOCq(*ja*+*n*-1) part of the local array *A* must contain the local pieces of the leading *ia*+*n*-1 by *ja*+*n*-1 part of the global matrix, and:

- If *uplo* = 'U', the leading  $n \times n$  upper triangular part of the global real symmetric or complex Hermitian submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  must contain the upper triangular part of the submatrix, and the strictly lower triangular part is not referenced.
- If *uplo* = 'L', the leading  $n \times n$  lower triangular part of the global real symmetric or complex Hermitian submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  must contain the lower triangular part of the submatrix, and the strictly upper triangular part is not referenced.

Scope: **local**

Specified as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 78 on page 449. Details about the square block-cyclic data distribution of global matrix *A* are stored in *desc\_a*.

*ia* is the row index of the global matrix *A*, identifying the first row of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq M\_A$  and  $ia+n-1 \leq M\_A$ .

*ja* is the column index of the global matrix *A*, identifying the first column of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$  and  $ja+n-1 \leq N\_A$ .

*desc\_a* is the array descriptor for global matrix *A*, described in the following table:

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_A < p$	Global

<i>desc_a</i>	Name	Description	Limits	Scope
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq \text{CSRC\_A} < q$	Global
9	LLD_A	The leading dimension of the local array	$\text{LLD\_A} \geq \max(1, \text{LOCp}(\text{M\_A}))$	Local

Specified as: an array of (at least) length 9, containing fullword integers.

*b* is the local part of the global general matrix *B*, containing the right-hand sides of the system. This identifies the **first element** of the local array *B*. This subroutine computes the location of the first element of the local subarray used, based on *ib*, *jb*, *desc\_b*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading  $\text{LOCp}(\text{ib}+n-1)$  by  $\text{LOCq}(\text{jb}+nrhs-1)$  part of the local array *B* must contain the local pieces of the leading  $\text{ib}+n-1$  by  $\text{jb}+nrhs-1$  part of the global matrix.

Scope: **local**

Specified as: an  $\text{LLD\_B}$  by (at least)  $\text{LOCq}(\text{N\_B})$  array, containing numbers of the data type indicated in Table 79 on page 458. Details about the block-cyclic data distribution of global matrix *B* are stored in *desc\_b*.

*ib* is the row index of the global matrix *B*, identifying the first row of the submatrix *B*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq \text{ib} \leq \text{M\_B}$  and  $\text{ib}+n-1 \leq \text{M\_B}$ .

*jb* is the column index of the global matrix *B*, identifying the first column of the submatrix *B*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq \text{jb} \leq \text{N\_B}$  and  $\text{jb}+nrhs-1 \leq \text{N\_B}$ .

*desc\_b* is the array descriptor for global matrix *B*, described in the following table:

<i>desc_b</i>	Name	Description	Limits	Scope
1	DTYPE_B	Descriptor type	$\text{DTYPE\_B}=1$	Global
2	CTXT_B	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_B	Number of rows in the global matrix	If $n = 0$ or $nrhs = 0$ : $\text{M\_B} \geq 0$ Otherwise: $\text{M\_B} \geq 1$	Global
4	N_B	Number of columns in the global matrix	If $n = 0$ or $nrhs = 0$ : $\text{N\_B} \geq 0$ Otherwise: $\text{N\_B} \geq 1$	Global
5	MB_B	Row block size	$\text{MB\_B} \geq 1$	Global
6	NB_B	Column block size	$\text{NB\_B} \geq 1$	Global
7	RSRC_B	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq \text{RSRC\_B} < p$	Global

<i>desc_b</i>	Name	Description	Limits	Scope
8	CSRC_B	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq \text{CSRC\_B} < q$	Global
9	LLD_B	The leading dimension of the local array	$\text{LLD\_B} \geq \max(1, \text{LOCp}(\text{M\_B}))$	Local

Specified as: an array of (at least) length 9, containing fullword integers.

*info* See On Return.

### On Return

*a* is the updated local part of the global matrix *A*, containing the results of the factorization.

Scope: **local**

Returned as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 78 on page 449.

*b* is the updated local part of the global matrix *B*, containing the solution vectors.

Scope: **local**

Returned as: an LLD\_B by (at least) LOCq(N\_B) array, containing numbers of the data type indicated in Table 79 on page 458.

*info* has the following meaning:

If *info* = 0, the global real symmetric or complex Hermitian submatrix *A* is positive definite, and the factorization and solve completed normally.

If *info* > 0, the leading minor of order *k* of the global real symmetric or complex Hermitian submatrix *A* is not positive definite. *info* is set equal to *k*, where the leading minor was encountered at  $A_{ia+k-1, ja+k-1}$ . The factorization is not completed. *A* is overwritten with the partial factors. The solution submatrix *B* is not computed.

Scope: **global**

Returned as: a fullword integer; *info* ≥ 0.

### Notes and Coding Rules

1. In your C program, argument *info* must be passed by reference.
2. If *n* > 0 and *nrhs* = 0, only the factorization is computed.
3. This subroutine accepts lowercase letters for the *uplo* argument.
4. On input to PZPOSV, the imaginary parts of the diagonal elements of the complex Hermitian matrix *A* are assumed to be zero, so you do not have to set these values. On output, they are set to zero.
5. The matrices must have no common elements; otherwise, results are unpredictable.
6. The way these subroutines handle nonpositive definiteness differs from ScaLAPACK. These subroutines use the *info* argument to provide information about the nonpositive definiteness of *A*, like ScaLAPACK, but also provides an error message.



7. The NUMROC utility subroutine can be used to determine the values of  $\text{LOCp}(M_)$  and  $\text{LOCq}(N_)$  used in the argument descriptions above. For details, see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28 and “NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process” on page 868.
8. For suggested block sizes, see “Coding Tips for Optimizing Parallel Performance” on page 77.
9. On both input and output, matrices  $A$  and  $B$  conform to ScaLAPACK format.
10. The following values must be equal:  $\text{CTXT\_A} = \text{CTXT\_B}$ .
11. The global real symmetric or complex Hermitian matrix  $A$  must be distributed using a square block-cyclic distribution; that is,  $\text{MB\_A} = \text{NB\_A}$ .
12. The following block sizes must be equal:  $\text{MB\_A} = \text{MB\_B}$ .
13. The global real symmetric or complex Hermitian matrix  $A$  must be aligned on a block row boundary; that is,  $ia-1$  must be a multiple of  $\text{MB\_A}$ .
14. The block row offset of  $A$  must be equal to the block column offset of  $A$ ; that is,  $\text{mod}(ia-1, \text{MB\_A}) = \text{mod}(ja-1, \text{NB\_A})$ .
15. The block row offset of  $A$  must be equal to the block row offset of  $B$ ; that is,  $\text{mod}(ia-1, \text{MB\_A}) = \text{mod}(ib-1, \text{MB\_B})$ .
16. In the process grid, the process row containing the first row of the submatrix  $A$  must also contain the first row of the submatrix  $B$ ; that is,  $iarow = ibrow$ , where:
  - $iarow = \text{mod}((((ia-1)/\text{MB\_A}) + \text{RSRC\_A}), p)$
  - $ibrow = \text{mod}((((ib-1)/\text{MB\_B}) + \text{RSRC\_B}), p)$
17. For the Parallel ESSL SMP libraries, these subroutines use nonblocking collective communications; therefore, the  $\text{MP\_SINGLE\_THREAD}$  environment variable must be set to **NO** (which is the default value). See the *IBM Parallel Environment for AIX: MPI Programming Guide* for more information.

## Error Conditions

### Computational Errors

Matrix  $A$  is not positive definite. For details, see the description of the *info* argument.

### Resource Errors

Unable to allocate work space

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1.  $\text{DTYPE\_A}$  is invalid.
2.  $\text{DTYPE\_B}$  is invalid.

#### Stage 2:

1.  $\text{CTXT\_A}$  is invalid.

#### Stage 3:

1. This subroutine was called from outside the process grid.

#### Stage 4:

1.  $uplo \neq 'U'$  or  $'L'$
2.  $n < 0$

3.  $nrhs < 0$
4.  $M\_A < 0$  and  $n = 0$ ;  $M\_A < 1$  otherwise
5.  $N\_A < 0$  and  $n = 0$ ;  $N\_A < 1$  otherwise
6.  $ia < 1$
7.  $ja < 1$
8.  $MB\_A < 1$
9.  $NB\_A < 1$
10.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
11.  $CSRC\_A < 0$  or  $CSRC\_A \geq q$
12.  $M\_B < 0$  and  $(n = 0$  or  $nrhs = 0)$ ;  $M\_B < 1$  otherwise
13.  $N\_B < 0$  and  $(n = 0$  or  $nrhs = 0)$ ;  $N\_B < 1$  otherwise
14.  $ib < 1$
15.  $jb < 1$
16.  $MB\_B < 1$
17.  $NB\_B < 1$
18.  $RSRC\_B < 0$  or  $RSRC\_B \geq p$
19.  $CSRC\_B < 0$  or  $CSRC\_B \geq q$
20.  $CTXT\_A \neq CTXT\_B$

**Stage 5:** If  $n \neq 0$ :

1.  $ia > M\_A$
2.  $ja > N\_A$
3.  $ia+n-1 > M\_A$
4.  $ja+n-1 > N\_A$

If  $n \neq 0$  and  $nrhs \neq 0$ :

5.  $ib > M\_B$
6.  $jb > N\_B$
7.  $ib+n-1 > M\_B$
8.  $jb+nrhs-1 > N\_B$

In all cases:

9.  $MB\_A \neq NB\_A$
10.  $\text{mod}(ia-1, MB\_A) \neq \text{mod}(ja-1, NB\_A)$
11.  $MB\_B \neq MB\_A$
12.  $\text{mod}(ia-1, MB\_A) \neq \text{mod}(ib-1, MB\_B)$ .
13.  $\text{mod}(ia-1, MB\_A) \neq 0$
14. In the process grid, the process row containing the first row of the submatrix *A* does not contain the first row of the submatrix *B*; that is,  $iarow \neq ibrow$ , where:
  - $iarow = \text{mod}(((ia-1)/MB\_A)+RSRC\_A), p)$
  - $ibrow = \text{mod}(((ib-1)/MB\_B)+RSRC\_B), p)$

**Stage 6:**

1.  $LLD\_A < \max(1, \text{LOCp}(M\_A))$
2.  $LLD\_B < \max(1, \text{LOCp}(M\_B))$

Each of the following global input arguments are checked to determine whether its value differs from the value specified on process  $P_{00}$ :

3. *uplo* differs.
4. *n* differs.
5. *nrhs* differs.
6. *ia* differs.
7. *ja* differs.
8. *DTYPE\_A* differs.
9. *M\_A* differs.
10. *N\_A* differs.
11. *MB\_A* differs.

12. NB\_A differs.
13. RSRC\_A differs.
14. CSRC\_A differs.
15. *ib* differs.
16. *jb* differs.
17. DTYPE\_B differs.
18. M\_B differs.
19. N\_B differs.
20. MB\_B differs.
21. NB\_B differs.
22. RSRC\_B differs.
23. CSRC\_B differs.

## Examples

### Example 1

This example solves the positive definite real symmetric system  $AX = B$  where  $A$  is a  $9 \times 9$  positive definite real symmetric matrix and  $B$  contains 5 right-hand sides using a  $2 \times 2$  process grid.

This example uses a global submatrix  $B$  within a global matrix  $B$  by specifying  $ib = 1$  and  $jb = 2$ .

By specifying  $CSRC\_B = 1$ , the columns of global matrix  $B$  are distributed over the process grid starting in the second column of the process grid.

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      UPLO  N  NRHS A  IA  JA  DESC_A  B  IB  JB  DESC_B  INFO
      |    |    |   |   |   |         |  |  |         |   |
CALL PDPOSV( 'L' , 9 , 5 , A , 1 , 1 , DESC_A , B , 1 , 2 , DESC_B , INFO )
```

	Desc_A	Desc_B
DTYPE_	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	9	9
N_	9	6
MB_	3	3
NB_	3	2
RSRC_	0	0
CSRC_	0	1
LLD_	See below <sup>2</sup>	See below <sup>2</sup>

#### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:

## PDPOSV and PZPOSV

```
LLD_A = MAX(1, NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_B = MAX(1, NUMROC(M_B, MB_B, MYROW, RSRC_B, NPROW))
```

In this example,  $LLD\_A = LLD\_B = 6$  on  $P_{00}$  and  $P_{01}$ , and  $LLD\_A = LLD\_B = 3$  on  $P_{10}$  and  $P_{11}$ .

Global real symmetric matrix  $A$  of order 9 with block size  $3 \times 3$ :

B,D	0	1	2
0	$\begin{bmatrix} 1.0 & . & . \\ 1.0 & 2.0 & . \\ 1.0 & 2.0 & 3.0 \end{bmatrix}$	$\begin{bmatrix} . & . & . \\ . & . & . \\ . & . & . \end{bmatrix}$	$\begin{bmatrix} . & . & . \\ . & . & . \\ . & . & . \end{bmatrix}$
1	$\begin{bmatrix} 1.0 & 2.0 & 3.0 \\ 1.0 & 2.0 & 3.0 \\ 1.0 & 2.0 & 3.0 \end{bmatrix}$	$\begin{bmatrix} 4.0 & . & . \\ 4.0 & 5.0 & . \\ 4.0 & 5.0 & 6.0 \end{bmatrix}$	$\begin{bmatrix} . & . & . \\ . & . & . \\ . & . & . \end{bmatrix}$
2	$\begin{bmatrix} 1.0 & 2.0 & 3.0 \\ 1.0 & 2.0 & 3.0 \\ 1.0 & 2.0 & 3.0 \end{bmatrix}$	$\begin{bmatrix} 4.0 & 5.0 & 6.0 \\ 4.0 & 5.0 & 6.0 \\ 4.0 & 5.0 & 6.0 \end{bmatrix}$	$\begin{bmatrix} 7.0 & . & . \\ 7.0 & 8.0 & . \\ 7.0 & 8.0 & 9.0 \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

Local arrays for  $A$ :

p,q	0	1
0	$\begin{bmatrix} 1.0 & . & . & . & . & . \\ 1.0 & 2.0 & . & . & . & . \\ 1.0 & 2.0 & 3.0 & . & . & . \\ 1.0 & 2.0 & 3.0 & 7.0 & . & . \\ 1.0 & 2.0 & 3.0 & 7.0 & 8.0 & . \\ 1.0 & 2.0 & 3.0 & 7.0 & 8.0 & 9.0 \end{bmatrix}$	$\begin{bmatrix} . & . & . \\ . & . & . \\ . & . & . \\ 4.0 & 5.0 & 6.0 \\ 4.0 & 5.0 & 6.0 \\ 4.0 & 5.0 & 6.0 \end{bmatrix}$
1	$\begin{bmatrix} 1.0 & 2.0 & 3.0 & . & . & . \\ 1.0 & 2.0 & 3.0 & . & . & . \\ 1.0 & 2.0 & 3.0 & . & . & . \end{bmatrix}$	$\begin{bmatrix} 4.0 & . & . \\ 4.0 & 5.0 & . \\ 4.0 & 5.0 & 6.0 \end{bmatrix}$

After the global matrix  $B$  is distributed over the process grid, only a portion of the global data structure is used—that is, global submatrix  $B$ . Following is the global  $9 \times 5$  submatrix  $B$ , starting at row 1 and column 2 in global general  $9 \times 6$  matrix  $B$  with block size  $3 \times 2$ :

B,D	0	1	2
0	$\begin{bmatrix} . & 18.0 \\ . & 34.0 \\ . & 48.0 \end{bmatrix}$	$\begin{bmatrix} 27.0 & 36.0 \\ 51.0 & 68.0 \\ 72.0 & 96.0 \end{bmatrix}$	$\begin{bmatrix} 45.0 & 9.0 \\ 85.0 & 17.0 \\ 120.0 & 24.0 \end{bmatrix}$
1	$\begin{bmatrix} . & 60.0 \\ . & 70.0 \\ . & 78.0 \end{bmatrix}$	$\begin{bmatrix} 90.0 & 120.0 \\ 105.0 & 140.0 \\ 117.0 & 156.0 \end{bmatrix}$	$\begin{bmatrix} 150.0 & 30.0 \\ 175.0 & 35.0 \\ 195.0 & 39.0 \end{bmatrix}$
2	$\begin{bmatrix} . & 84.0 \\ . & 88.0 \\ . & 90.0 \end{bmatrix}$	$\begin{bmatrix} 126.0 & 168.0 \\ 132.0 & 176.0 \\ 135.0 & 180.0 \end{bmatrix}$	$\begin{bmatrix} 210.0 & 42.0 \\ 220.0 & 44.0 \\ 225.0 & 45.0 \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	1	0 2
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

**Note:** The first column of  $B$  begins in the second column of the process grid.

Local arrays for  $B$ :

p,q	0	1
0	27.0 36.0 51.0 68.0 72.0 96.0 126.0 168.0 132.0 176.0 135.0 180.0	. 18.0 45.0 9.0 . 34.0 85.0 17.0 . 48.0 120.0 24.0 . 84.0 210.0 42.0 . 88.0 220.0 44.0 . 90.0 225.0 45.0
1	90.0 120.0 105.0 140.0 117.0 156.0	. 60.0 150.0 30.0 . 70.0 175.0 35.0 . 78.0 195.0 39.0

**Output:**

Global real symmetric matrix  $A$  of order 9 with block size  $3 \times 3$ :

B,D	0	1	2
0	1.0 . . 1.0 1.0 . 1.0 1.0 1.0	. . . . . . . . .	. . . . . . . . .
1	1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0	1.0 . . 1.0 1.0 . 1.0 1.0 1.0	. . . . . . . . .
2	1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0	1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0	1.0 . . 1.0 1.0 . 1.0 1.0 1.0

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

Local arrays for  $A$ :

p,q	0	1
0	1.0 . . . . . 1.0 1.0 . . . . 1.0 1.0 1.0 . . . 1.0 1.0 1.0 1.0 . . 1.0 1.0 1.0 1.0 1.0 . 1.0 1.0 1.0 1.0 1.0 1.0	. . . . . . . . . 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1	1.0 1.0 1.0 . . . 1.0 1.0 1.0 . . . 1.0 1.0 1.0 . . .	1.0 . . 1.0 1.0 . 1.0 1.0 1.0

After the global matrix  $B$  is distributed over the process grid, only a portion of the global data structure is used—that is, global submatrix  $B$ . Following is the global  $9 \times 5$  submatrix  $B$ , starting at row 1 and column 2 in global general  $9 \times 6$  matrix  $B$  with block size  $3 \times 2$ :

B,D	0	1	2
0	$\begin{bmatrix} . & 2.0 \\ . & 2.0 \\ . & 2.0 \end{bmatrix}$	$\begin{bmatrix} 3.0 & 4.0 \\ 3.0 & 4.0 \\ 3.0 & 4.0 \end{bmatrix}$	$\begin{bmatrix} 5.0 & 1.0 \\ 5.0 & 1.0 \\ 5.0 & 1.0 \end{bmatrix}$
1	$\begin{bmatrix} . & 2.0 \\ . & 2.0 \\ . & 2.0 \end{bmatrix}$	$\begin{bmatrix} 3.0 & 4.0 \\ 3.0 & 4.0 \\ 3.0 & 4.0 \end{bmatrix}$	$\begin{bmatrix} 5.0 & 1.0 \\ 5.0 & 1.0 \\ 5.0 & 1.0 \end{bmatrix}$
2	$\begin{bmatrix} . & 2.0 \\ . & 2.0 \\ . & 2.0 \end{bmatrix}$	$\begin{bmatrix} 3.0 & 4.0 \\ 3.0 & 4.0 \\ 3.0 & 4.0 \end{bmatrix}$	$\begin{bmatrix} 5.0 & 1.0 \\ 5.0 & 1.0 \\ 5.0 & 1.0 \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	1	0 2
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

**Note:** The first column of  $B$  begins in the second column of the process grid.

Local arrays for  $B$ :

p,q	0	1
0	$\begin{bmatrix} 3.0 & 4.0 \\ 3.0 & 4.0 \\ 3.0 & 4.0 \\ 3.0 & 4.0 \\ 3.0 & 4.0 \\ 3.0 & 4.0 \end{bmatrix}$	$\begin{bmatrix} . & 2.0 & 5.0 & 1.0 \\ . & 2.0 & 5.0 & 1.0 \\ . & 2.0 & 5.0 & 1.0 \\ . & 2.0 & 5.0 & 1.0 \\ . & 2.0 & 5.0 & 1.0 \\ . & 2.0 & 5.0 & 1.0 \end{bmatrix}$
1	$\begin{bmatrix} 3.0 & 4.0 \\ 3.0 & 4.0 \\ 3.0 & 4.0 \end{bmatrix}$	$\begin{bmatrix} . & 2.0 & 5.0 & 1.0 \\ . & 2.0 & 5.0 & 1.0 \\ . & 2.0 & 5.0 & 1.0 \end{bmatrix}$

The value of *info* is 0 on all processes.

## Example 2

This example solves the positive definite complex Hermitian system  $AX = B$  where  $A$  is a  $9 \times 9$  positive definite complex Hermitian matrix and  $B$  contains 5 right-hand sides using a  $2 \times 2$  process grid.

This example uses a global submatrix  $B$  within a global matrix  $B$  by specifying  $ib = 1$  and  $jb = 2$ .

By specifying  $CSRC\_B = 1$ , the columns of global matrix  $B$  are distributed over the process grid starting in the second column of the process grid.

## Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
```

```
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)
```

```

      UPLD  N  NRHS  A  IA  JA  DESC_A  B  IB  JB  DESC_B  INFO
      |    |    |    |    |    |    |    |    |    |    |
CALL PZPOSV( 'L' , 9 , 5 , A , 1 , 1 , DESC_A , B , 1 , 2 , DESC_B , INFO )

```

	Desc_A	Desc_B
DTYPE_	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	9	9
N_	9	6
MB_	3	3
NB_	3	2
RSRC_	0	0
CSRC_	0	1
LLD_	See below <sup>2</sup>	See below <sup>2</sup>

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:

```

LLD_A = MAX(1,NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_B = MAX(1,NUMROC(M_B, MB_B, MYROW, RSRC_B, NPROW))

```

In this example, LLD\_A = LLD\_B = 6 on P<sub>00</sub> and P<sub>01</sub>, and  
LLD\_A = LLD\_B = 3 on P<sub>10</sub> and P<sub>11</sub>.

Global complex Hermitian matrix *A* of order 9 with block size 3 × 3:

B,D	0	1	2
0	<div> <div>(18.0, 0.0)</div> <div>(1.0, 1.0) (18.0, 0.0)</div> <div>(1.0, 1.0) (3.0, 1.0) (18.0, 0.0)</div> </div>	<div> <div>.</div> <div>.</div> <div>.</div> </div>	<div> <div>.</div> <div>.</div> <div>.</div> </div>
1	<div> <div>(1.0, 1.0) (3.0, 1.0) (5.0, 1.0)</div> <div>(1.0, 1.0) (3.0, 1.0) (5.0, 1.0)</div> <div>(1.0, 1.0) (3.0, 1.0) (5.0, 1.0)</div> </div>	<div> <div>(18.0, 0.0)</div> <div>(7.0, 1.0) (18.0, 0.0)</div> <div>(7.0, 1.0) (9.0, 1.0) (18.0, 0.0)</div> </div>	<div> <div>.</div> <div>.</div> <div>.</div> </div>
2	<div> <div>(1.0, 1.0) (3.0, 1.0) (5.0, 1.0)</div> <div>(1.0, 1.0) (3.0, 1.0) (5.0, 1.0)</div> <div>(1.0, 1.0) (3.0, 1.0) (5.0, 1.0)</div> </div>	<div> <div>(7.0, 1.0) (9.0, 1.0) (11.0, 1.0)</div> <div>(7.0, 1.0) (9.0, 1.0) (11.0, 1.0)</div> <div>(7.0, 1.0) (9.0, 1.0) (11.0, 1.0)</div> </div>	<div> <div>(18.0, 0.0)</div> <div>(13.0, 1.0) (18.0, 0.0)</div> <div>(13.0, 1.0) (15.0, 1.0) (18.0, 0.0)</div> </div>

**Note:** On input, the imaginary parts of the diagonal elements of the complex Hermitian matrix *A* are assumed to be zero, so you do not have to set these values.

The following is the 2 × 2 process grid:

B,D	0 2	1
0	P <sub>00</sub>	P <sub>01</sub>
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for *A*:

## PDPOSV and PZPOSV

p,q	0						1		
0	(18.0, .)	.	.	.	.	.	.	.	.
	(1.0, 1.0)	(18.0, .)	.	.	.	.	.	.	.
	(1.0, 1.0)	(3.0, 1.0)	(18.0, .)	.	.	.	.	.	.
	(1.0, 1.0)	(3.0, 1.0)	(5.0, 1.0)	(18.0, .)	.	.	(7.0, 1.0)	(9.0, 1.0)	(11.0, 1.0)
	(1.0, 1.0)	(3.0, 1.0)	(5.0, 1.0)	(13.0, 1.0)	(18.0, .)	.	(7.0, 1.0)	(9.0, 1.0)	(11.0, 1.0)
	(1.0, 1.0)	(3.0, 1.0)	(5.0, 1.0)	(13.0, 1.0)	(15.0, 1.0)	(18.0, .)	(7.0, 1.0)	(9.0, 1.0)	(11.0, 1.0)
1	(1.0, 1.0)	(3.0, 1.0)	(5.0, 1.0)	.	.	.	(18.0, .)	.	.
	(1.0, 1.0)	(3.0, 1.0)	(5.0, 1.0)	.	.	.	(7.0, 1.0)	(18.0, .)	.
	(1.0, 1.0)	(3.0, 1.0)	(5.0, 1.0)	.	.	.	(7.0, 1.0)	(9.0, 1.0)	(18.0, .)

After the global matrix  $B$  is distributed over the process grid, only a portion of the global data structure is used—that is, global submatrix  $B$ . Following is the global  $9 \times 5$  submatrix  $B$ , starting at row 1 and column 2 in global general  $9 \times 6$  matrix  $B$  with block size  $3 \times 2$ :

B,D	0	1	2
0	(60.0, 10.0)	(86.0, 2.0)	(112.0, -6.0)
	(86.0, 28.0)	(126.0, 22.0)	(166.0, 16.0)
	(108.0, 44.0)	(160.0, 40.0)	(212.0, 36.0)
1	(126.0, 58.0)	(188.0, 56.0)	(250.0, 54.0)
	(140.0, 70.0)	(210.0, 70.0)	(280.0, 70.0)
	(150.0, 80.0)	(226.0, 82.0)	(302.0, 84.0)
2	(156.0, 88.0)	(236.0, 92.0)	(316.0, 96.0)
	(158.0, 94.0)	(240.0, 100.0)	(322.0, 106.0)
	(156.0, 98.0)	(238.0, 106.0)	(320.0, 114.0)

The following is the  $2 \times 2$  process grid:

B,D	1	0 2
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

**Note:** The first column of  $B$  begins in the second column of the process grid.

Local arrays for  $B$ :

p,q	0	1
0	(86.0, 2.0)	(112.0, -6.0)
	(126.0, 22.0)	(166.0, 16.0)
	(160.0, 40.0)	(212.0, 36.0)
	(236.0, 92.0)	(316.0, 96.0)
	(240.0, 100.0)	(322.0, 106.0)
	(238.0, 106.0)	(320.0, 114.0)
1	(188.0, 56.0)	(250.0, 54.0)
	(210.0, 70.0)	(280.0, 70.0)
	(226.0, 82.0)	(302.0, 84.0)

**Output:**

Global complex Hermitian matrix  $A$  of order 9 with block size  $3 \times 3$ :



B,D	0	1	2
0	$\begin{pmatrix} (4.2, 0.0) & \cdot & \cdot \\ (0.24, 0.24) & (4.2, 0.0) & \cdot \\ (0.24, 0.24) & (0.68, 0.24) & (4.2, 0.0) \end{pmatrix}$	$\begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix}$	$\begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix}$
1	$\begin{pmatrix} (0.24, 0.24) & (0.68, 0.24) & (1.1, 0.24) \\ (0.24, 0.24) & (0.68, 0.24) & (1.1, 0.24) \\ (0.24, 0.24) & (0.68, 0.24) & (1.1, 0.24) \end{pmatrix}$	$\begin{pmatrix} (4.0, 0.0) & \cdot & \cdot \\ (1.3, 0.25) & (3.8, 0.0) & \cdot \\ (1.3, 0.25) & (1.4, 0.26) & (3.5, 0.0) \end{pmatrix}$	$\begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix}$
2	$\begin{pmatrix} (0.24, 0.24) & (0.68, 0.24) & (1.1, 0.24) \\ (0.24, 0.24) & (0.68, 0.24) & (1.1, 0.24) \\ (0.24, 0.24) & (0.68, 0.24) & (1.1, 0.24) \end{pmatrix}$	$\begin{pmatrix} (1.3, 0.25) & (1.4, 0.26) & (1.5, 0.28) \\ (1.3, 0.25) & (1.4, 0.26) & (1.5, 0.28) \\ (1.3, 0.25) & (1.4, 0.26) & (1.5, 0.28) \end{pmatrix}$	$\begin{pmatrix} (3.2, 0.0) & \cdot & \cdot \\ (1.6, 0.32) & (2.7, 0.0) & \cdot \\ (1.6, 0.32) & (1.6, 0.37) & (2.2, 0.0) \end{pmatrix}$

**Note:** On output, the imaginary parts of the diagonal elements of the matrix are set to zero.

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

Local arrays for  $A$ :

p,q	0	1
0	$\begin{pmatrix} (4.2, 0.0) & \cdot & \cdot & \cdot & \cdot & \cdot \\ (0.24, 0.24) & (4.2, 0.0) & \cdot & \cdot & \cdot & \cdot \\ (0.24, 0.24) & (0.68, 0.24) & (4.2, 0.0) & \cdot & \cdot & \cdot \\ (0.24, 0.24) & (0.68, 0.24) & (1.1, 0.24) & (3.2, 0.0) & \cdot & \cdot \\ (0.24, 0.24) & (0.68, 0.24) & (1.1, 0.24) & (1.6, 0.32) & (2.7, 0.0) & \cdot \\ (0.24, 0.24) & (0.68, 0.24) & (1.1, 0.24) & (1.6, 0.32) & (1.6, 0.37) & (2.2, 0.0) \end{pmatrix}$	$\begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ (1.3, 0.25) & (1.4, 0.26) & (1.5, 0.28) \\ (1.3, 0.25) & (1.4, 0.26) & (1.5, 0.28) \\ (1.3, 0.25) & (1.4, 0.26) & (1.5, 0.28) \end{pmatrix}$
1	$\begin{pmatrix} (0.24, 0.24) & (0.68, 0.24) & (1.1, 0.24) & \cdot & \cdot & \cdot \\ (0.24, 0.24) & (0.68, 0.24) & (1.1, 0.24) & \cdot & \cdot & \cdot \\ (0.24, 0.24) & (0.68, 0.24) & (1.1, 0.24) & \cdot & \cdot & \cdot \end{pmatrix}$	$\begin{pmatrix} (4.0, 0.0) & \cdot & \cdot \\ (1.3, 0.25) & (3.8, 0.0) & \cdot \\ (1.3, 0.25) & (1.4, 0.26) & (3.5, 0.0) \end{pmatrix}$

After the global matrix  $B$  is distributed over the process grid, only a portion of the global data structure is used—that is, global submatrix  $B$ . Following is the global  $9 \times 5$  submatrix  $B$ , starting at row 1 and column 2 in global general  $9 \times 6$  matrix  $B$  with block size  $3 \times 2$ :

B,D	0	1	2
0	$\begin{pmatrix} \cdot & (2.0, 1.0) \\ \cdot & (2.0, 1.0) \\ \cdot & (2.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (3.0, 1.0) & (4.0, 1.0) \\ (3.0, 1.0) & (4.0, 1.0) \\ (3.0, 1.0) & (4.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (5.0, 1.0) & (1.0, 1.0) \\ (5.0, 1.0) & (1.0, 1.0) \\ (5.0, 1.0) & (1.0, 1.0) \end{pmatrix}$
1	$\begin{pmatrix} \cdot & (2.0, 1.0) \\ \cdot & (2.0, 1.0) \\ \cdot & (2.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (3.0, 1.0) & (4.0, 1.0) \\ (3.0, 1.0) & (4.0, 1.0) \\ (3.0, 1.0) & (4.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (5.0, 1.0) & (1.0, 1.0) \\ (5.0, 1.0) & (1.0, 1.0) \\ (5.0, 1.0) & (1.0, 1.0) \end{pmatrix}$
2	$\begin{pmatrix} \cdot & (2.0, 1.0) \\ \cdot & (2.0, 1.0) \\ \cdot & (2.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (3.0, 1.0) & (4.0, 1.0) \\ (3.0, 1.0) & (4.0, 1.0) \\ (3.0, 1.0) & (4.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (5.0, 1.0) & (1.0, 1.0) \\ (5.0, 1.0) & (1.0, 1.0) \\ (5.0, 1.0) & (1.0, 1.0) \end{pmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	1	0 2
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

**Note:** The first column of  $B$  begins in the second column of the process grid.

Local arrays for  $B$ :

p,q	0		1			
0	(3.0, 4.0)	(3.0, 4.0)	.	(2.0, 1.0)	(5.0, 1.0)	(1.0, 1.0)
	(3.0, 4.0)	(3.0, 4.0)	.	(2.0, 1.0)	(5.0, 1.0)	(1.0, 1.0)
	(3.0, 4.0)	(3.0, 4.0)	.	(2.0, 1.0)	(5.0, 1.0)	(1.0, 1.0)
	(3.0, 4.0)	(3.0, 4.0)	.	(2.0, 1.0)	(5.0, 1.0)	(1.0, 1.0)
	(3.0, 4.0)	(3.0, 4.0)	.	(2.0, 1.0)	(5.0, 1.0)	(1.0, 1.0)
	(3.0, 4.0)	(3.0, 4.0)	.	(2.0, 1.0)	(5.0, 1.0)	(1.0, 1.0)
1	(3.0, 4.0)	(3.0, 4.0)	.	(2.0, 1.0)	(5.0, 1.0)	(1.0, 1.0)
	(3.0, 4.0)	(3.0, 4.0)	.	(2.0, 1.0)	(5.0, 1.0)	(1.0, 1.0)
	(3.0, 4.0)	(3.0, 4.0)	.	(2.0, 1.0)	(5.0, 1.0)	(1.0, 1.0)

The value of *info* is 0 on all processes.

## PDPOTRF and PZPOTRF — Positive Definite Real Symmetric or Complex Hermitian Matrix Factorization

### Purpose

PDPOTRF uses Cholesky factorization to factor a positive definite real symmetric matrix  $A$  into one of the following forms:

- $A = LL^T$  if  $uplo = 'L'$ .
- $A = U^T U$  if  $uplo = 'U'$ .

PZPOTRF uses Cholesky factorization to factor a positive definite complex Hermitian matrix  $A$  into one of the following forms:

- $A = LL^H$  if  $uplo = 'L'$ .
- $A = U^H U$  if  $uplo = 'U'$ .

In the formulas above:

- $A$  represents the global positive definite real symmetric or complex Hermitian submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  to be factored.
- $L$  is a lower triangular matrix.
- $U$  is an upper triangular matrix.

To solve the system of equations with any number of right-hand sides, follow the call to these subroutines with one or more calls to PDPOTRS or PZPOTRS, respectively.

If  $n = 0$ , no computation is performed and the subroutine returns after doing some parameter checking. See references [17], [19], [23], [38], and [39].

Table 78. Data Types

$A$	Subroutine
Long-precision real	PDPOTRF
Long-precision complex	PZPOTRF

### Syntax

<b>Fortran</b>	CALL PDPOTRF   PZPOTRF ( <i>uplo</i> , <i>n</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>info</i> )
<b>C and C++</b>	pdpotrf   pzpotr ( <i>uplo</i> , <i>n</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>info</i> );

### On Entry

*uplo* indicates whether the upper or lower triangular part of the global real symmetric or complex Hermitian submatrix  $A$  is referenced, where:

If  $uplo = 'U'$ , the upper triangular part is referenced.

If  $uplo = 'L'$ , the lower triangular part is referenced.

Scope: **global**

Specified as: a single character;  $uplo = 'U'$  or  $'L'$ .

*n* is the number of rows and columns in submatrix  $A$  used in the computation.

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

*a* is the local part of the global real symmetric or complex Hermitian matrix *A*, used in the system of equations. This identifies the **first element** of the local array *A*. This subroutine computes the location of the first element of the local subarray used, based on *ia*, *ja*, *desc\_a*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading LOCp(*ia*+*n*-1) by LOCq(*ja*+*n*-1) part of the local array *A* must contain the local pieces of the leading *ia*+*n*-1 by *ja*+*n*-1 part of the global matrix, and:

- If *uplo* = 'U', the leading  $n \times n$  upper triangular part of the global real symmetric or complex Hermitian submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  must contain the upper triangular part of the submatrix, and the strictly lower triangular part is not referenced.
- If *uplo* = 'L', the leading  $n \times n$  lower triangular part of the global real symmetric or complex Hermitian submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  must contain the lower triangular part of the submatrix, and the strictly upper triangular part is not referenced.

Scope: **local**

Specified as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 78 on page 449. Details about the square block-cyclic data distribution of global matrix *A* are stored in *desc\_a*.

*ia* is the row index of the global matrix *A*, identifying the first row of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq M\_A$  and  $ia+n-1 \leq M\_A$ .

*ja* is the column index of the global matrix *A*, identifying the first column of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$  and  $ja+n-1 \leq N\_A$ .

*desc\_a* is the array descriptor for global matrix *A*, described in the following table:

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_A < p$	Global

<i>desc_a</i>	Name	Description	Limits	Scope
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq \text{CSRC\_A} < q$	Global
9	LLD_A	The leading dimension of the local array	$\text{LLD\_A} \geq \max(1, \text{LOCp}(\text{M\_A}))$	Local

Specified as: an array of (at least) length 9, containing fullword integers.

*info* See On Return.

### On Return

*a* is the updated local part of the global matrix *A*, containing the results of the factorization.

Scope: **local**

Returned as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 78 on page 449.

*info* has the following meaning:

If *info* = 0, global real symmetric or complex Hermitian submatrix *A* is positive definite, and the factorization completed normally.

If *info* > 0, the leading minor of order *k* of the global real symmetric or complex Hermitian submatrix *A* is not positive definite. *info* is set equal to *k*, where the leading minor was encountered at  $A_{ia+k-1, ja+k-1}$ . The factorization is not completed. *A* is overwritten with the partial factors.

Scope: **global**

Returned as: a fullword integer; *info* ≥ 0.

## Notes and Coding Rules

1. In your C program, argument *info* must be passed by reference.
2. This subroutine accepts lowercase letters for the *uplo* argument.
3. On input to PZPOTRF, the imaginary parts of the diagonal elements of the complex Hermitian matrix *A* are assumed to be zero, so you do not have to set these values. On output, they are set to zero.
4. The scalar data specified for input argument *n* must be the same for both PDPOTRF/PZPOTRF and PDPOTRS/PZPOTRS.
5. The global submatrix *A* input to PDPOTRS/PZPOTRS must be the same as for the corresponding output argument for PDPOTRF/PZPOTRF; and thus, the scalar data specified for *ia*, *ja*, and the contents of *desc\_a* must also be the same.
6. The NUMROC utility subroutine can be used to determine the values of LOCp(M\_) and LOCq(N\_) used in the argument descriptions above. For details, see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28 and “NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process” on page 868.
7. The way these subroutines handle nonpositive definiteness differs from ScaLAPACK. These subroutines use the *info* argument to provide information about the nonpositive definiteness of *A*, like ScaLAPACK, but also provide an error message.

8. On both input and output, matrix  $A$  conforms to ScaLAPACK format.
9. The global real symmetric or complex Hermitian matrix  $A$  must be distributed using a square block-cyclic distribution; that is,  $MB\_A = NB\_A$ .
10. The global real symmetric or complex Hermitian matrix  $A$  must be aligned on a block row boundary; that is,  $ia-1$  must be a multiple of  $MB\_A$ .
11. The block row offset of  $A$  must be equal to the block column offset of  $A$ ; that is,  $\text{mod}(ia-1, MB\_A) = \text{mod}(ja-1, NB\_A)$ .
12. For the Parallel ESSL SMP libraries, these subroutines use nonblocking collective communications; therefore, the `MP_SINGLE_THREAD` environment variable must be set to `NO` (which is the default value). See the *IBM Parallel Environment for AIX: MPI Programming Guide* for more information.

## Performance Considerations

1. For suggested block sizes, see “Coding Tips for Optimizing Parallel Performance” on page 77.
2. For optimal performance, you should use a square process grid to minimize the communication path length in both directions.
3. For optimal performance, take the following items into consideration when choosing the  $NB\_A$  ( $= MB\_A$ ) value:
  - Whether you are using the upper or lower triangular part of  $A$ , which may affect performance.
  - The cache size of the computational nodes.  $NB\_A$  determines the granularity of the most expensive part of the computation, which tends to increase the optimal value of  $NB\_A$ .
  - The communication and synchronization overhead. This has two aspects, the cost of internal synchronization points and the cost of broadcasts. These tend to slightly decrease the optimal value of  $NB\_A$ .
  - The model of communication adapter you are using.
  - Load balancing. For the best processor utilization, it is necessary for the processor nodes to be active for as long as possible; therefore, each one should have as many blocks as possible. For a given problem size, this tends to decrease the optimal value of  $NB\_A$  (best load balancing: 1) and is most relevant at very small problem sizes.
  - If  $NB\_A$  is equal to a power of 2, performance may be degraded.
  - Use the following rules of thumb for reasonably-sized problems:
    - For the SERIAL processors, choose  $NB\_A$  in the following range:
      - For PDPOTRF, use [30, 50], avoiding 32.
      - For PZPOTRF, use [10, 25], avoiding 16.
    - For the SMP processors, choose  $NB\_A$  in the following range:
      - For PDPOTRF, use [70, 100].
      - For PZPOTRF, use [30, 50], avoiding 32.

## Error Conditions

### Computational Errors

Matrix  $A$  is not positive definite. For details, see the description of the *info* argument.

### Resource Errors

Unable to allocate work space

## Input-Argument and Miscellaneous Errors

### Stage 1:

1. DTYPE\_A is invalid.

### Stage 2:

1. CTXT\_A is invalid.

### Stage 3:

1. This subroutine was called from outside the process grid.

### Stage 4:

1.  $uplo \neq 'U' \text{ or } 'L'$
2.  $n < 0$
3.  $M\_A < 0$  and  $n = 0$ ;  $M\_A < 1$  otherwise
4.  $N\_A < 0$  and  $n = 0$ ;  $N\_A < 1$  otherwise
5.  $ia < 1$
6.  $ja < 1$
7.  $MB\_A < 1$
8.  $NB\_A < 1$
9.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
10.  $CSRC\_A < 0$  or  $CSRC\_A \geq q$

### Stage 5: If $n \neq 0$ :

1.  $ia > M\_A$
2.  $ja > N\_A$
3.  $ia+n-1 > M\_A$
4.  $ja+n-1 > N\_A$

In all cases:

5.  $MB\_A \neq NB\_A$
6.  $\text{mod}(ia-1, MB\_A) \neq \text{mod}(ja-1, NB\_A)$
7.  $\text{mod}(ia-1, MB\_A) \neq 0$

### Stage 6:

1.  $LLD\_A < \max(1, \text{LOCp}(M\_A))$

Each of the following global input arguments are checked to determine whether its value differs from the value specified on process  $P_{00}$ :

2.  $uplo$  differs.
3.  $n$  differs.
4.  $ia$  differs.
5.  $ja$  differs.
6. DTYPE\_A differs.
7. M\_A differs.
8. N\_A differs.
9. MB\_A differs.
10. NB\_A differs.
11. RSRC\_A differs.
12. CSRC\_A differs.

## Examples

### Example 1

This example factors a  $9 \times 9$  positive definite real symmetric matrix using a  $2 \times 2$  process grid.

### Call Statements and Input:

## PDPOTRF and PZPOTRF

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)
```

```

      UPLO   N   A   IA   JA   DESC_A   INFO
      |     |   |   |   |   |         |
CALL PDPOTRF( 'L' , 9 , A , 1 , 1 , DESC_A , INFO )

```

	Desc_A
DTYPE_	1
CTXT_	<i>icontxt</i> <sup>1</sup>
M_	9
N_	9
MB_	3
NB_	3
RSRC_	0
CSRC_	0
LLD_	See below <sup>2</sup>

### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.

2. Each process should set the LLD\_ as follows:

```
LLD_A = MAX(1, NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
```

In this example, LLD\_A = 6 on P<sub>00</sub> and P<sub>01</sub>, and LLD\_A = 3 on P<sub>10</sub> and P<sub>11</sub>.

Global real symmetric matrix A of order 9 with block size 3 × 3:

B,D	0	1	2
0	$\begin{bmatrix} 1.0 & . & . \\ 1.0 & 2.0 & . \\ 1.0 & 2.0 & 3.0 \end{bmatrix}$	$\begin{bmatrix} . & . & . \\ . & . & . \\ . & . & . \end{bmatrix}$	$\begin{bmatrix} . & . & . \\ . & . & . \\ . & . & . \end{bmatrix}$
1	$\begin{bmatrix} 1.0 & 2.0 & 3.0 \\ 1.0 & 2.0 & 3.0 \\ 1.0 & 2.0 & 3.0 \end{bmatrix}$	$\begin{bmatrix} 4.0 & . & . \\ 4.0 & 5.0 & . \\ 4.0 & 5.0 & 6.0 \end{bmatrix}$	$\begin{bmatrix} . & . & . \\ . & . & . \\ . & . & . \end{bmatrix}$
2	$\begin{bmatrix} 1.0 & 2.0 & 3.0 \\ 1.0 & 2.0 & 3.0 \\ 1.0 & 2.0 & 3.0 \end{bmatrix}$	$\begin{bmatrix} 4.0 & 5.0 & 6.0 \\ 4.0 & 5.0 & 6.0 \\ 4.0 & 5.0 & 6.0 \end{bmatrix}$	$\begin{bmatrix} 7.0 & . & . \\ 7.0 & 8.0 & . \\ 7.0 & 8.0 & 9.0 \end{bmatrix}$

The following is the 2 × 2 process grid:

B,D	0 2	1
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for A:

p,q	0	1
	1.0 . . . . .	. . .



0	1.0	2.0	.	.	.	.	.	.
	1.0	2.0	3.0	.	.	.	.	.
	1.0	2.0	3.0	7.0	.	.	.	.
	1.0	2.0	3.0	7.0	8.0	.	.	.
	1.0	2.0	3.0	7.0	8.0	9.0	.	.
<hr/>								
1	1.0	2.0	3.0	.	.	.	4.0	.
	1.0	2.0	3.0	.	.	.	4.0	5.0
	1.0	2.0	3.0	.	.	.	4.0	5.0 6.0

### Output:

Global real symmetric matrix  $A$  of order 9 with block size  $3 \times 3$ :

B,D	0	1	2
0	1.0 . .	. . .	. . .
	1.0 1.0 .	. . .	. . .
	1.0 1.0 1.0	. . .	. . .
<hr/>			
1	1.0 1.0 1.0	1.0 . .	. . .
	1.0 1.0 1.0	1.0 1.0 .	. . .
	1.0 1.0 1.0	1.0 1.0 1.0	. . .
<hr/>			
2	1.0 1.0 1.0	1.0 1.0 1.0	1.0 . .
	1.0 1.0 1.0	1.0 1.0 1.0	1.0 1.0 .
	1.0 1.0 1.0	1.0 1.0 1.0	1.0 1.0 1.0

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
<hr/>		
0	$P_{00}$	$P_{01}$
2		
<hr/>		
1	$P_{10}$	$P_{11}$

Local arrays for  $A$ :

p,q	0	1
0	1.0 . .	. . .
	1.0 1.0 .	. . .
	1.0 1.0 1.0	. . .
	1.0 1.0 1.0 1.0	1.0 1.0 1.0
	1.0 1.0 1.0 1.0 1.0	1.0 1.0 1.0
	1.0 1.0 1.0 1.0 1.0 1.0	1.0 1.0 1.0
<hr/>		
1	1.0 1.0 1.0 . . .	1.0 . .
	1.0 1.0 1.0 . . .	1.0 1.0 .
	1.0 1.0 1.0 . . .	1.0 1.0 1.0

The value of *info* is 0 on all processes.

### Example 2

This example factors a  $9 \times 9$  positive definite complex Hermitian matrix using a  $2 \times 2$  process grid.

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)
```

## PDPOTRF and PZPOTRF

```

          UPLO   N   A   IA   JA   DESC_A   INFO
          |     |   |   |   |   |         |
CALL PZPOTRF( 'L' , 9 , A , 1 , 1 , DESC_A , INFO )

```

	Desc_A
DTYPE_	1
CTXT_	<i>icontxt</i> <sup>1</sup>
M_	9
N_	9
MB_	3
NB_	3
RSRC_	0
CSRC_	0
LLD_	See below <sup>2</sup>

### Notes:

- icontxt* is the output of the BLACS\_GRIDINIT call.
- Each process should set the LLD\_ as follows:  

$$\text{LLD\_A} = \text{MAX}(1, \text{NUMROC}(\text{M\_A}, \text{MB\_A}, \text{MYROW}, \text{RSRC\_A}, \text{NPROW}))$$
In this example, LLD\_A = 6 on P<sub>00</sub> and P<sub>01</sub>, and LLD\_A = 3 on P<sub>10</sub> and P<sub>11</sub>.

Global complex Hermitian matrix *A* of order 9 with block size 3 × 3:

B,D	0	1	2
0	(18.0, 0.0) . . (1.0, 1.0) (18.0, 0.0) . (1.0, 1.0) (3.0, 1.0) (18.0, 0.0)	. . . . . . . . .	. . . . . . . . .
1	(1.0, 1.0) (3.0, 1.0) (5.0, 1.0) (1.0, 1.0) (3.0, 1.0) (5.0, 1.0) (1.0, 1.0) (3.0, 1.0) (5.0, 1.0)	(18.0, 0.0) . . (7.0, 1.0) (18.0, 0.0) . (7.0, 1.0) (9.0, 1.0) (18.0, 0.0)	. . . . . . . . .
2	(1.0, 1.0) (3.0, 1.0) (5.0, 1.0) (1.0, 1.0) (3.0, 1.0) (5.0, 1.0) (1.0, 1.0) (3.0, 1.0) (5.0, 1.0)	(7.0, 1.0) (9.0, 1.0) (11.0, 1.0) (7.0, 1.0) (9.0, 1.0) (11.0, 1.0) (7.0, 1.0) (9.0, 1.0) (11.0, 1.0)	(18.0, 0.0) . . (13.0, 1.0) (18.0, 0.0) . (13.0, 1.0) (15.0, 1.0) (18.0, 0.0)

**Note:** On input, the imaginary parts of the diagonal elements of the complex Hermitian matrix *A* are assumed to be zero, so you do not have to set these values.

The following is the 2 × 2 process grid:

B,D	0 2	1
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for *A*:

p,q	0						1		
0	(18.0, . )	.	.	.	.	.	.	.	.
	(1.0, 1.0)	(18.0, . )	.	.	.	.	.	.	.
	(1.0, 1.0)	(3.0, 1.0)	(18.0, . )	.	.	.	.	.	.
	(1.0, 1.0)	(3.0, 1.0)	(5.0, 1.0)	(18.0, . )	.	.	(7.0, 1.0)	(9.0, 1.0)	(11.0, 1.0)
	(1.0, 1.0)	(3.0, 1.0)	(5.0, 1.0)	(13.0, 1.0)	(18.0, . )	.	(7.0, 1.0)	(9.0, 1.0)	(11.0, 1.0)
	(1.0, 1.0)	(3.0, 1.0)	(5.0, 1.0)	(13.0, 1.0)	(15.0, 1.0)	(18.0, . )	(7.0, 1.0)	(9.0, 1.0)	(11.0, 1.0)
1	(1.0, 1.0)	(3.0, 1.0)	(5.0, 1.0)	.	.	.	(18.0, . )	.	.
	(1.0, 1.0)	(3.0, 1.0)	(5.0, 1.0)	.	.	.	(7.0, 1.0)	(18.0, . )	.
	(1.0, 1.0)	(3.0, 1.0)	(5.0, 1.0)	.	.	.	(7.0, 1.0)	(9.0, 1.0)	(18.0, . )

### Output:

Global complex Hermitian matrix  $A$  of order 9 with block size  $3 \times 3$ :

B,D	0			1			2		
0	(4.2, 0.0)	.	.	.	.	.	.	.	.
1	(0.24, 0.24)	(4.2, 0.0)	.	.	.	.	.	.	.
	(0.24, 0.24)	(0.68, 0.24)	(4.2, 0.0)	.	.	.	.	.	.
	(0.24, 0.24)	(0.68, 0.24)	(1.1, 0.24)	(4.0, 0.0)	.	.	.	.	.
2	(0.24, 0.24)	(0.68, 0.24)	(1.1, 0.24)	(1.3, 0.25)	(3.8, 0.0)	.	.	.	.
	(0.24, 0.24)	(0.68, 0.24)	(1.1, 0.24)	(1.3, 0.25)	(1.4, 0.26)	(3.5, 0.0)	.	.	.
	(0.24, 0.24)	(0.68, 0.24)	(1.1, 0.24)	(1.3, 0.25)	(1.4, 0.26)	(1.5, 0.28)	(3.2, 0.0)	.	.
3	(0.24, 0.24)	(0.68, 0.24)	(1.1, 0.24)	(1.3, 0.25)	(1.4, 0.26)	(1.5, 0.28)	(1.6, 0.32)	(2.7, 0.0)	.
	(0.24, 0.24)	(0.68, 0.24)	(1.1, 0.24)	(1.3, 0.25)	(1.4, 0.26)	(1.5, 0.28)	(1.6, 0.32)	(1.6, 0.37)	(2.2, 0.0)
	(0.24, 0.24)	(0.68, 0.24)	(1.1, 0.24)	(1.3, 0.25)	(1.4, 0.26)	(1.5, 0.28)	(1.6, 0.32)	(1.6, 0.37)	(2.2, 0.0)

**Note:** On output, the imaginary parts of the diagonal elements of the matrix are set to zero.

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

Local arrays for  $A$ :

p,q	0						1		
0	(4.2, 0.0)	.	.	.	.	.	.	.	.
	(0.24, 0.24)	(4.2, 0.0)	.	.	.	.	.	.	.
	(0.24, 0.24)	(0.68, 0.24)	(4.2, 0.0)	.	.	.	.	.	.
	(0.24, 0.24)	(0.68, 0.24)	(1.1, 0.24)	(3.2, 0.0)	.	.	(1.3, 0.25)	(1.4, 0.26)	(1.5, 0.28)
	(0.24, 0.24)	(0.68, 0.24)	(1.1, 0.24)	(1.6, 0.32)	(2.7, 0.0)	.	(1.3, 0.25)	(1.4, 0.26)	(1.5, 0.28)
	(0.24, 0.24)	(0.68, 0.24)	(1.1, 0.24)	(1.6, 0.32)	(1.6, 0.37)	(2.2, 0.0)	(1.3, 0.25)	(1.4, 0.26)	(1.5, 0.28)
1	(0.24, 0.24)	(0.68, 0.24)	(1.1, 0.24)	.	.	.	(4.0, 0.0)	.	.
	(0.24, 0.24)	(0.68, 0.24)	(1.1, 0.24)	.	.	.	(1.3, 0.25)	(3.8, 0.0)	.
	(0.24, 0.24)	(0.68, 0.24)	(1.1, 0.24)	.	.	.	(1.3, 0.25)	(1.4, 0.26)	(3.5, 0.0)

The value of *info* is 0 on all processes.

## PDPOTRS and PZPOTRS — Positive Definite Real Symmetric or Complex Hermitian Matrix Solve

### Purpose

These subroutines solve the following systems of equations for multiple right-hand sides:

- $AX = B$

where, in the formula above:

- $A$  represents the global positive definite real symmetric or complex Hermitian submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  factored by Cholesky factorization.
- $B$  represents the global general submatrix  $B_{ib:ib+n-1, jb:jb+nrhs-1}$  containing the right-hand sides in its columns.
- $X$  represents the global general submatrix  $B_{ib:ib+n-1, jb:jb+nrhs-1}$  containing the solution vectors in its columns.

This subroutine uses the results of the factorization of matrix  $A$ , produced by a preceding call to PDPOTRF or PZPOTRF, respectively. For details on the factorization, see “PDPOTRF and PZPOTRF — Positive Definite Real Symmetric or Complex Hermitian Matrix Factorization” on page 449.

If  $n = 0$  or  $nrhs = 0$ , no computation is performed and the subroutine returns after doing some parameter checking. See references [17], [19], [23], [38], and [39].

Table 79. Data Types

$A, B$	Subroutine
Long-precision real	PDPOTRS
Long-precision complex	PZPOTRS

### Syntax

<b>Fortran</b>	CALL PDPOTRS   PZPOTRS ( <i>uplo</i> , <i>n</i> , <i>nrhs</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>b</i> , <i>ib</i> , <i>jb</i> , <i>desc_b</i> , <i>info</i> )
<b>C and C++</b>	pdpotrs   pzpots ( <i>uplo</i> , <i>n</i> , <i>nrhs</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>b</i> , <i>ib</i> , <i>jb</i> , <i>desc_b</i> , <i>info</i> );

### On Entry

*uplo* indicates whether the upper or lower triangular part of the global real symmetric or complex Hermitian submatrix  $A$  is referenced, where:

If *uplo* = 'U', the upper triangular part is referenced.

If *uplo* = 'L', the lower triangular part is referenced.

Scope: **global**

Specified as: a single character; *uplo* = 'U' or 'L'.

*n* is the order of the factored submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

*nrhs* is the number of right-hand sides— that is, the number of columns in submatrix  $B$  used in the computation.

Scope: **global**

Specified as: a fullword integer;  $nrhs \geq 0$ .

*a* is the local part of the global real symmetric or complex Hermitian matrix *A*, containing the factorization of matrix *A* produced by a preceding call to PDPOTRF or PZPOTRF, respectively. This identifies the **first element** of the local array *A*. This subroutine computes the location of the first element of the local subarray used, based on *ia*, *ja*, *desc\_a*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading LOCp(*ia*+*n*-1) by LOCq(*ja*+*n*-1) part of the local array *A* must contain the local pieces of the leading *ia*+*n*-1 by *ja*+*n*-1 part of the global matrix, and:

- If *uplo* = 'U', the leading  $n \times n$  upper triangular part of the global real symmetric or complex Hermitian submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  must contain the upper triangular part of the submatrix, and the strictly lower triangular part is not referenced.
- If *uplo* = 'L', the leading  $n \times n$  lower triangular part of the global real symmetric or complex Hermitian submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  must contain the lower triangular part of the submatrix, and the strictly upper triangular part is not referenced.

Scope: **local**

Specified as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 79 on page 458. Details about the square block-cyclic data distribution of global matrix *A* are stored in *desc\_a*.

*ia* is the row index of the global matrix *A*, identifying the first row of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq M\_A$  and  $ia+n-1 \leq M\_A$ .

*ja* is the column index of the global matrix *A*, identifying the first column of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$  and  $ja+n-1 \leq N\_A$ .

*desc\_a* is the array descriptor for global matrix *A*, described in the following table:

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_A < p$	Global

## PDPOTRS and PZPOTRS

<i>desc_a</i>	Name	Description	Limits	Scope
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq \text{CSRC\_A} < q$	Global
9	LLD_A	The leading dimension of the local array	$\text{LLD\_A} \geq \max(1, \text{LOCp}(\text{M\_A}))$	Local

Specified as: an array of (at least) length 9, containing fullword integers.

*b* is the local part of the global general matrix *B*, containing the right-hand sides of the system. This identifies the **first element** of the local array *B*. This subroutine computes the location of the first element of the local subarray used, based on *ib*, *jb*, *desc\_b*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading  $\text{LOCp}(\text{ib}+n-1)$  by  $\text{LOCq}(\text{jb}+nrhs-1)$  part of the local array *B* must contain the local pieces of the leading  $\text{ib}+n-1$  by  $\text{jb}+nrhs-1$  part of the global matrix.

Scope: **local**

Specified as: an  $\text{LLD\_B}$  by (at least)  $\text{LOCq}(\text{N\_B})$  array, containing numbers of the data type indicated in Table 79 on page 458. Details about the block-cyclic data distribution of global matrix *B* are stored in *desc\_b*.

*ib* is the row index of the global matrix *B*, identifying the first row of the submatrix *B*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq \text{ib} \leq \text{M\_B}$  and  $\text{ib}+n-1 \leq \text{M\_B}$ .

*jb* is the column index of the global matrix *B*, identifying the first column of the submatrix *B*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq \text{jb} \leq \text{N\_B}$  and  $\text{jb}+nrhs-1 \leq \text{N\_B}$ .

*desc\_b* is the array descriptor for global matrix *B*, described in the following table:

<i>desc_b</i>	Name	Description	Limits	Scope
1	DTYPE_B	Descriptor type	$\text{DTYPE\_B}=1$	Global
2	CTXT_B	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_B	Number of rows in the global matrix	If $n = 0$ or $nrhs = 0$ : $\text{M\_B} \geq 0$ Otherwise: $\text{M\_B} \geq 1$	Global
4	N_B	Number of columns in the global matrix	If $n = 0$ or $nrhs = 0$ : $\text{N\_B} \geq 0$ Otherwise: $\text{N\_B} \geq 1$	Global
5	MB_B	Row block size	$\text{MB\_B} \geq 1$	Global
6	NB_B	Column block size	$\text{NB\_B} \geq 1$	Global
7	RSRC_B	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq \text{RSRC\_B} < p$	Global

<i>desc_b</i>	Name	Description	Limits	Scope
8	CSRC_B	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq \text{CSRC\_B} < q$	Global
9	LLD_B	The leading dimension of the local array	$\text{LLD\_B} \geq \max(1, \text{LOCp}(\text{M\_B}))$	Local

Specified as: an array of (at least) length 9, containing fullword integers.

*info* See On Return.

### On Return

*b* is the updated local part of the global matrix *B*, containing the solution vectors.

Scope: **local**

Returned as: an LLD\_B by (at least) LOCq(N\_B) array, containing numbers of the data type indicated in Table 79 on page 458.

*info* indicates that a successful computation occurred.

Scope: **global**

Returned as: a fullword integer; *info* = 0.

## Notes and Coding Rules

1. In your C program, argument *info* must be passed by reference.
2. This subroutine accepts lowercase letters for the *uplo* argument.
3. The matrices must have no common elements; otherwise, results are unpredictable.
4. The scalar data specified for input argument *n* must be the same for both PDPOTRF/PZPOTRF and PDPOTRS/PZPOTRS.
5. The global submatrix *A* input to PDPOTRS/PZPOTRS must be the same as for the corresponding output argument for PDPOTRF/PZPOTRS; and thus, the scalar data specified for *ia*, *ja*, and the contents of *desc\_a* must also be the same.
6. The NUMROC utility subroutine can be used to determine the values of LOCp(M\_) and LOCq(N\_) used in the argument descriptions above. For details, see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28 and “NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process” on page 868.
7. For suggested block sizes, see “Coding Tips for Optimizing Parallel Performance” on page 77.
8. On both input and output, matrices *A* and *B* conform to ScaLAPACK format.
9. The following values must be equal: CTXT\_A = CTXT\_B.
10. The global real symmetric or complex Hermitian matrix *A* must be distributed using a square block-cyclic distribution; that is, MB\_A = NB\_A.
11. The following block sizes must be equal: MB\_A = MB\_B.
12. The global real symmetric or complex Hermitian matrix *A* must be aligned on a block row boundary; that is, *ia*–1 must be a multiple of MB\_A.

13. The block row offset of  $A$  must be equal to the block column offset of  $A$ ; that is,  $\text{mod}(ia-1, MB\_A) = \text{mod}(ja-1, NB\_A)$ .
14. The block row offset of  $A$  must be equal to the block row offset of  $B$ ; that is,  $\text{mod}(ia-1, MB\_A) = \text{mod}(ib-1, MB\_B)$ .
15. In the process grid, the process row containing the first row of the submatrix  $A$  must also contain the first row of the submatrix  $B$ ; that is,  $iarow = ibrow$ , where:
  - $iarow = \text{mod}((((ia-1)/MB\_A)+RSRC\_A), p)$
  - $ibrow = \text{mod}((((ib-1)/MB\_B)+RSRC\_B), p)$

## Error Conditions

### Computational Errors

None

**Note:** If the factorization performed by PDPOTRF/PZPOTRF failed because of a nonpositive definite matrix  $A$ , the results returned by this subroutine are unpredictable. For details, see the *info* output argument for PDPOTRF/PZPOTRF.

### Resource Errors

Unable to allocate work space

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. DTYPE\_A is invalid.
2. DTYPE\_B is invalid.

#### Stage 2:

1. CTEXT\_A is invalid.

#### Stage 3:

1. This subroutine was called from outside the process grid.

#### Stage 4:

1.  $uplo \neq 'U'$  or  $'L'$
2.  $n < 0$
3.  $nrhs < 0$
4.  $M\_A < 0$  and  $n = 0$ ;  $M\_A < 1$  otherwise
5.  $N\_A < 0$  and  $n = 0$ ;  $N\_A < 1$  otherwise
6.  $ia < 1$
7.  $ja < 1$
8.  $MB\_A < 1$
9.  $NB\_A < 1$
10.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
11.  $CSRC\_A < 0$  or  $CSRC\_A \geq q$
12.  $M\_B < 0$  and  $(n = 0$  or  $nrhs = 0)$ ;  $M\_B < 1$  otherwise
13.  $N\_B < 0$  and  $(n = 0$  or  $nrhs = 0)$ ;  $N\_B < 1$  otherwise
14.  $ib < 1$
15.  $jb < 1$
16.  $MB\_B < 1$
17.  $NB\_B < 1$
18.  $RSRC\_B < 0$  or  $RSRC\_B \geq p$
19.  $CSRC\_B < 0$  or  $CSRC\_B \geq q$



20. CTXT\_A  $\neq$  CTXT\_B

**Stage 5:** If  $n \neq 0$ :

1.  $ia > M\_A$
2.  $ja > N\_A$
3.  $ia+n-1 > M\_A$
4.  $ja+n-1 > N\_A$

If  $n \neq 0$  and  $nrhs \neq 0$ :

5.  $ib > M\_B$
6.  $jb > N\_B$
7.  $ib+nrhs-1 > M\_B$
8.  $jb+nrhs-1 > N\_B$

In all cases:

9.  $MB\_A \neq NB\_A$
10.  $\text{mod}(ia-1, MB\_A) \neq \text{mod}(ja-1, NB\_A)$
11.  $MB\_B \neq MB\_A$
12.  $\text{mod}(ia-1, MB\_A) \neq \text{mod}(ib-1, MB\_B)$ .
13.  $\text{mod}(ia-1, MB\_A) \neq 0$
14. In the process grid, the process row containing the first row of the submatrix *A* does not contain the first row of the submatrix *B*; that is,  $iarow \neq ibrow$ , where:
  - $iarow = \text{mod}((((ia-1)/MB\_A)+RSRC\_A), p)$
  - $ibrow = \text{mod}((((ib-1)/MB\_B)+RSRC\_B), p)$

**Stage 6:**

1.  $LLD\_A < \max(1, LOCp(M\_A))$
2.  $LLD\_B < \max(1, LOCp(M\_B))$

Each of the following global input arguments are checked to determine whether its value differs from the value specified on process  $P_{00}$ :

3. *uplo* differs.
4. *n* differs.
5. *nrhs* differs.
6. *ia* differs.
7. *ja* differs.
8. DTYPE\_A differs.
9. M\_A differs.
10. N\_A differs.
11. MB\_A differs.
12. NB\_A differs.
13. RSRC\_A differs.
14. CSRC\_A differs.
15. *ib* differs.
16. *jb* differs.
17. DTYPE\_B differs.
18. M\_B differs.
19. N\_B differs.
20. MB\_B differs.
21. NB\_B differs.
22. RSRC\_B differs.
23. CSRC\_B differs.

## Examples

### Example 1

This example solves the positive definite real symmetric system  $AX = B$  with 5 right-hand sides using a  $2 \times 2$  process grid. The transformed matrix  $A$  is the output from “Example 1” on page 453.

This example uses a global submatrix  $B$  within a global matrix  $B$  by specifying  $ib = 1$  and  $jb = 2$ .

By specifying  $CSRC\_B = 1$ , the columns of global matrix  $B$  are distributed over the process grid starting in the second column of the process grid.

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

          UPLO  N  NRHS A  IA  JA  DESC_A  B  IB  JB  DESC_B  INFO
          |    |    |   |   |   |      |  |  |  |      |   |
CALL PDPOTRS( 'L' , 9 , 5 , A , 1 , 1 , DESC_A , B , 1 , 2 , DESC_B , INFO )
```

	Desc_A	Desc_B
DTYPE_	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	9	9
N_	9	6
MB_	3	3
NB_	3	2
RSRC_	0	0
CSRC_	0	1
LLD_	See below <sup>2</sup>	See below <sup>2</sup>

#### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.

2. Each process should set the LLD\_ as follows:

```
LLD_A = MAX(1, NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_B = MAX(1, NUMROC(M_B, MB_B, MYROW, RSRC_B, NPROW))
```

In this example,  $LLD\_A = LLD\_B = 6$  on  $P_{00}$  and  $P_{01}$ , and  
 $LLD\_A = LLD\_B = 3$  on  $P_{10}$  and  $P_{11}$ .

After the global matrix  $B$  is distributed over the process grid, only a portion of the global data structure is used—that is, global submatrix  $B$ . Following is the global  $9 \times 5$  submatrix  $B$ , starting at row 1 and column 2 in global general  $9 \times 6$  matrix  $B$  with block size  $3 \times 2$ :

```
B,D      0      1      2
0  [  .  18.0 | 27.0 36.0 | 45.0  9.0 ]
    [  .  34.0 | 51.0 68.0 | 85.0 17.0 ]
    [  .  48.0 | 72.0 96.0 |120.0 24.0 ]
    -----|-----|-----
```

1	.	60.0	90.0	120.0	150.0	30.0
	.	70.0	105.0	140.0	175.0	35.0
	.	78.0	117.0	156.0	195.0	39.0
2	.	84.0	126.0	168.0	210.0	42.0
	.	88.0	132.0	176.0	220.0	44.0
	.	90.0	135.0	180.0	225.0	45.0

The following is the  $2 \times 2$  process grid:

B,D	1	0 2
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>

**Note:** The first column of  $B$  begins in the second column of the process grid.

Local arrays for  $B$ :

p,q	0	1
0	27.0 36.0 51.0 68.0 72.0 96.0 126.0 168.0 132.0 176.0 135.0 180.0	. 18.0 45.0 9.0 . 34.0 85.0 17.0 . 48.0 120.0 24.0 . 84.0 210.0 42.0 . 88.0 220.0 44.0 . 90.0 225.0 45.0
1	90.0 120.0 105.0 140.0 117.0 156.0	. 60.0 150.0 30.0 . 70.0 175.0 35.0 . 78.0 195.0 39.0

**Output:**

After the global matrix  $B$  is distributed over the process grid, only a portion of the global data structure is used—that is, global submatrix  $B$ . Following is the global  $9 \times 5$  submatrix  $B$ , starting at row 1 and column 2 in global general  $9 \times 6$  matrix  $B$  with block size  $3 \times 2$ :

B,D	0	1	2
0	. 2.0 . 2.0 . 2.0	3.0 4.0 3.0 4.0 3.0 4.0	5.0 1.0 5.0 1.0 5.0 1.0
1	. 2.0 . 2.0 . 2.0	3.0 4.0 3.0 4.0 3.0 4.0	5.0 1.0 5.0 1.0 5.0 1.0
2	. 2.0 . 2.0 . 2.0	3.0 4.0 3.0 4.0 3.0 4.0	5.0 1.0 5.0 1.0 5.0 1.0

The following is the  $2 \times 2$  process grid:

B,D	1	0 2
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>

**Note:** The first column of  $B$  begins in the second column of the process grid.

Local arrays for  $B$ :

p,q	0	1
0	3.0 4.0	. 2.0 5.0 1.0
	3.0 4.0	. 2.0 5.0 1.0
	3.0 4.0	. 2.0 5.0 1.0
	3.0 4.0	. 2.0 5.0 1.0
	3.0 4.0	. 2.0 5.0 1.0
	3.0 4.0	. 2.0 5.0 1.0
1	3.0 4.0	. 2.0 5.0 1.0
	3.0 4.0	. 2.0 5.0 1.0
	3.0 4.0	. 2.0 5.0 1.0

The value of *info* is 0 on all processes.

## Example 2

This example solves the positive definite complex Hermitian system  $AX = B$  with 5 right-hand sides using a  $2 \times 2$  process grid. The transformed matrix  $A$  is the output from “Example 2” on page 455.

This example uses a global submatrix  $B$  within a global matrix  $B$  by specifying  $ib = 1$  and  $jb = 2$ .

By specifying  $CSRC\_B = 1$ , the columns of global matrix  $B$  are distributed over the process grid starting in the second column of the process grid.

## Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      UPLO  N  NRHS  A  IA  JA  DESC_A  B  IB  JB  DESC_B  INFO
      |    |    |    |  |  |  |      |  |  |  |      |
CALL PZPOTRS( 'L' , 9 , 5 , A , 1 , 1 , DESC_A , B , 1 , 2 , DESC_B , INFO )
```

	Desc_A	Desc_B
DTYPE_	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	9	9
N_	9	6
MB_	3	3
NB_	3	2
RSRC_	0	0
CSRC_	0	1
LLD_	See below <sup>2</sup>	See below <sup>2</sup>

## Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:

LLD\_A = MAX(1, NUMROC(M\_A, MB\_A, MYROW, RSRC\_A, NPROW))  
 LLD\_B = MAX(1, NUMROC(M\_B, MB\_B, MYROW, RSRC\_B, NPROW))

In this example, LLD\_A = LLD\_B = 6 on P<sub>00</sub> and P<sub>01</sub>, and  
 LLD\_A = LLD\_B = 3 on P<sub>10</sub> and P<sub>11</sub>.

After the global matrix  $B$  is distributed over the process grid, only a portion of the global data structure is used—that is, global submatrix  $B$ . Following is the global  $9 \times 5$  submatrix  $B$ , starting at row 1 and column 2 in global general  $9 \times 6$  matrix  $B$  with block size  $3 \times 2$ :

B,D	0	1	2
0	$\begin{bmatrix} \cdot & (60.0, 10.0) \\ \cdot & (86.0, 28.0) \\ \cdot & (108.0, 44.0) \end{bmatrix}$	$\begin{bmatrix} (86.0, 2.0) & (112.0, -6.0) \\ (126.0, 22.0) & (166.0, 16.0) \\ (160.0, 40.0) & (212.0, 36.0) \end{bmatrix}$	$\begin{bmatrix} (138.0, -14.0) & (34.0, 18.0) \\ (206.0, 10.0) & (46.0, 34.0) \\ (264.0, 32.0) & (56.0, 48.0) \end{bmatrix}$
1	$\begin{bmatrix} \cdot & (126.0, 58.0) \\ \cdot & (140.0, 70.0) \\ \cdot & (150.0, 80.0) \end{bmatrix}$	$\begin{bmatrix} (188.0, 56.0) & (250.0, 54.0) \\ (210.0, 70.0) & (280.0, 70.0) \\ (226.0, 82.0) & (302.0, 84.0) \end{bmatrix}$	$\begin{bmatrix} (312.0, 52.0) & (64.0, 60.0) \\ (350.0, 70.0) & (70.0, 70.0) \\ (378.0, 86.0) & (74.0, 78.0) \end{bmatrix}$
2	$\begin{bmatrix} \cdot & (156.0, 88.0) \\ \cdot & (158.0, 94.0) \\ \cdot & (156.0, 98.0) \end{bmatrix}$	$\begin{bmatrix} (236.0, 92.0) & (316.0, 96.0) \\ (240.0, 100.0) & (322.0, 106.0) \\ (238.0, 106.0) & (320.0, 114.0) \end{bmatrix}$	$\begin{bmatrix} (396.0, 100.0) & (76.0, 84.0) \\ (404.0, 112.0) & (76.0, 88.0) \\ (402.0, 122.0) & (74.0, 90.0) \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	1	0 2
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>

**Note:** The first column of  $B$  begins in the second column of the process grid.

Local arrays for  $B$ :

p,q	0	1
0	$\begin{bmatrix} (86.0, 2.0) & (112.0, -6.0) \\ (126.0, 22.0) & (166.0, 16.0) \\ (160.0, 40.0) & (212.0, 36.0) \\ (236.0, 92.0) & (316.0, 96.0) \\ (240.0, 100.0) & (322.0, 106.0) \\ (238.0, 106.0) & (320.0, 114.0) \end{bmatrix}$	$\begin{bmatrix} \cdot & (60.0, 10.0) & (138.0, -14.0) & (34.0, 18.0) \\ \cdot & (86.0, 28.0) & (206.0, 10.0) & (46.0, 34.0) \\ \cdot & (108.0, 44.0) & (264.0, 32.0) & (56.0, 48.0) \\ \cdot & (156.0, 88.0) & (396.0, 100.0) & (76.0, 84.0) \\ \cdot & (158.0, 94.0) & (404.0, 112.0) & (76.0, 88.0) \\ \cdot & (156.0, 98.0) & (402.0, 122.0) & (74.0, 90.0) \end{bmatrix}$
1	$\begin{bmatrix} (188.0, 56.0) & (250.0, 54.0) \\ (210.0, 70.0) & (280.0, 70.0) \\ (226.0, 82.0) & (302.0, 84.0) \end{bmatrix}$	$\begin{bmatrix} \cdot & (126.0, 58.0) & (312.0, 52.0) & (64.0, 60.0) \\ \cdot & (140.0, 70.0) & (350.0, 70.0) & (70.0, 70.0) \\ \cdot & (150.0, 80.0) & (378.0, 86.0) & (74.0, 78.0) \end{bmatrix}$

**Output:**

After the global matrix  $B$  is distributed over the process grid, only a portion of the global data structure is used—that is, global submatrix  $B$ . Following is the global  $9 \times 5$  submatrix  $B$ , starting at row 1 and column 2 in global general  $9 \times 6$  matrix  $B$  with block size  $3 \times 2$ :

## PDPOTRS and PZPOTRS

B,D	0	1	2
0	<div> <div>.</div> <div>(2.0, 1.0)</div> </div> <div> <div>.</div> <div>(2.0, 1.0)</div> </div> <div> <div>.</div> <div>(2.0, 1.0)</div> </div>	<div> <div>(3.0, 1.0)</div> <div>(4.0, 1.0)</div> </div> <div> <div>(3.0, 1.0)</div> <div>(4.0, 1.0)</div> </div> <div> <div>(3.0, 1.0)</div> <div>(4.0, 1.0)</div> </div>	<div> <div>(5.0, 1.0)</div> <div>(1.0, 1.0)</div> </div> <div> <div>(5.0, 1.0)</div> <div>(1.0, 1.0)</div> </div> <div> <div>(5.0, 1.0)</div> <div>(1.0, 1.0)</div> </div>
1	<div> <div>.</div> <div>(2.0, 1.0)</div> </div> <div> <div>.</div> <div>(2.0, 1.0)</div> </div> <div> <div>.</div> <div>(2.0, 1.0)</div> </div>	<div> <div>(3.0, 1.0)</div> <div>(4.0, 1.0)</div> </div> <div> <div>(3.0, 1.0)</div> <div>(4.0, 1.0)</div> </div> <div> <div>(3.0, 1.0)</div> <div>(4.0, 1.0)</div> </div>	<div> <div>(5.0, 1.0)</div> <div>(1.0, 1.0)</div> </div> <div> <div>(5.0, 1.0)</div> <div>(1.0, 1.0)</div> </div> <div> <div>(5.0, 1.0)</div> <div>(1.0, 1.0)</div> </div>
2	<div> <div>.</div> <div>(2.0, 1.0)</div> </div> <div> <div>.</div> <div>(2.0, 1.0)</div> </div> <div> <div>.</div> <div>(2.0, 1.0)</div> </div>	<div> <div>(3.0, 1.0)</div> <div>(4.0, 1.0)</div> </div> <div> <div>(3.0, 1.0)</div> <div>(4.0, 1.0)</div> </div> <div> <div>(3.0, 1.0)</div> <div>(4.0, 1.0)</div> </div>	<div> <div>(5.0, 1.0)</div> <div>(1.0, 1.0)</div> </div> <div> <div>(5.0, 1.0)</div> <div>(1.0, 1.0)</div> </div> <div> <div>(5.0, 1.0)</div> <div>(1.0, 1.0)</div> </div>

The following is the  $2 \times 2$  process grid:

B,D	1	0 2
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

**Note:** The first column of  $B$  begins in the second column of the process grid.

Local arrays for  $B$ :

p,q	0	1
0	<div> <div>(3.0, 4.0)</div> <div>(3.0, 4.0)</div> </div> <div> <div>(3.0, 4.0)</div> <div>(3.0, 4.0)</div> </div> <div> <div>(3.0, 4.0)</div> <div>(3.0, 4.0)</div> </div> <div> <div>(3.0, 4.0)</div> <div>(3.0, 4.0)</div> </div> <div> <div>(3.0, 4.0)</div> <div>(3.0, 4.0)</div> </div> <div> <div>(3.0, 4.0)</div> <div>(3.0, 4.0)</div> </div>	<div> <div>.</div> <div>(2.0, 1.0)</div> </div> <div> <div>.</div> <div>(2.0, 1.0)</div> </div> <div> <div>.</div> <div>(2.0, 1.0)</div> </div> <div> <div>.</div> <div>(2.0, 1.0)</div> </div> <div> <div>.</div> <div>(2.0, 1.0)</div> </div> <div> <div>.</div> <div>(2.0, 1.0)</div> </div>
1	<div> <div>(3.0, 4.0)</div> <div>(3.0, 4.0)</div> </div> <div> <div>(3.0, 4.0)</div> <div>(3.0, 4.0)</div> </div> <div> <div>(3.0, 4.0)</div> <div>(3.0, 4.0)</div> </div>	<div> <div>.</div> <div>(2.0, 1.0)</div> </div> <div> <div>.</div> <div>(2.0, 1.0)</div> </div> <div> <div>.</div> <div>(2.0, 1.0)</div> </div>

The value of *info* is 0 on all processes.

## PDPOTRI and PZPOTRI — Positive Definite Real Symmetric or Complex Hermitian Matrix Inverse

### Purpose

PDPOTRI and PZPOTRI compute the inverse of a positive definite real symmetric or complex Hermitian matrix  $A$ . These subroutines use the results of the factorization of matrix  $A$ , produced by a preceding call to PDPOTRF or PZPOTRF, respectively. For details on the factorization, see “PDPOTRF and PZPOTRF — Positive Definite Real Symmetric or Complex Hermitian Matrix Factorization” on page 449.

If  $n = 0$ , no computation is performed and the subroutine returns after doing some parameter checking.

Table 80. Data Types

$A$	Subroutine
Long-precision real	PDPOTRI
Long-precision complex	PZPOTRI

### Syntax

<b>Fortran</b>	CALL PDPOTRI   PZPOTRI ( <i>uplo</i> , <i>n</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>info</i> )
<b>C and C++</b>	pdpotri   pzpotri ( <i>uplo</i> , <i>n</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>info</i> );

### On Entry

*uplo* indicates whether the upper or lower triangular part of the global submatrix  $A$  is referenced, where:

If *uplo* = 'U', the upper triangular part is referenced.

If *uplo* = 'L', the lower triangular part is referenced.

Scope: **global**

Specified as: a single character; *uplo* = 'U' or 'L'.

*n* is the order of the factored submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

*a* is the local part of the global real symmetric or complex Hermitian matrix  $A$ , containing the factorization of matrix  $A$  produced by a preceding call to PDPOTRF or PZPOTRF, respectively. This identifies the **first element** of the local array  $A$ . This subroutine computes the location of the first element of the local subarray used, based on *ia*, *ja*, *desc\_a*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading LOCp(*ia*+*n*−1) by LOCq(*ja*+*n*−1) part of the local array  $A$  must contain the local pieces of the leading *ia*+*n*−1 by *ja*+*n*−1 part of the global matrix, and:

- If *uplo* = 'U', the leading  $n \times n$  upper triangular part of the global submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  must contain the upper triangular part of the submatrix, and the strictly lower triangular part is not referenced.

- If  $uplo = 'L'$ , the leading  $n \times n$  lower triangular part of the global submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  must contain the lower triangular part of the submatrix, and the strictly upper triangular part is not referenced.

Scope: **local**

Specified as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 80 on page 469. Details about the square block-cyclic data distribution of global matrix  $A$  are stored in  $desc_a$ .

$ia$  is the row index of the global matrix  $A$ , identifying the first row of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq M\_A$  and  $ia+n-1 \leq M\_A$ .

$ja$  is the column index of the global matrix  $A$ , identifying the first column of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$  and  $ja+n-1 \leq N\_A$ .

$desc_a$  is the array descriptor for global matrix  $A$ , described in the following table:

$desc\_a$	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_A < p$	Global
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < q$	Global
9	LLD_A	The leading dimension of the local array	$LLD\_A \geq \max(1, LOCp(M\_A))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

$info$  See On Return.

### On Return

$a$  is the updated local part of the global matrix  $A$ , containing the inverse of  $A$ .



Scope: **local**

Returned as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 80 on page 469.

*info* has the following meaning:

If *info* = 0, global submatrix *A* is positive definite and the inverse completed normally.

If *info* > 0, global submatrix *A* is not positive definite and the inverse could not be computed. For *info* = *k*, the corresponding diagonal element of  $\mathbf{U}_{k, k}$  or  $\mathbf{L}_{k, k}$  is exactly zero.

Scope: **global**

Returned as: a fullword integer; *info* ≥ 0.

## Notes and Coding Rules

1. In your C program, argument *info* must be passed by reference.
2. This subroutine accepts lowercase letters for the *uplo* argument.
3. The scalar data specified for input argument *n* must be the same for both PDPOTRF/PZPOTRF and PDPOTRI/PZPOTRI.
4. The global submatrix for *A* input to PDPOTRI/PZPOTRI must be the same as for the corresponding output arguments for PDPOTRF/PZPOTRF; and thus, the scalar data specified for *ia*, *ja*, and the contents of *desc\_a* must also be the same.
5. The NUMROC utility subroutine can be used to determine the values of LOCp(M\_) and LOCq(N\_) used in the argument descriptions above. For details, see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28 and “NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process” on page 868.
6. The way these subroutines handle nonpositive definiteness differs from ScaLAPACK. These subroutines use the *info* argument to provide information about the nonpositive definiteness of *A*, like ScaLAPACK, but also provide an error message.
7. For suggested block sizes, see “Coding Tips for Optimizing Parallel Performance” on page 77.
8. On both input and output, matrix *A* conforms to ScaLAPACK format.
9. The global matrix *A* must be distributed using a square block-cyclic distribution; that is, MB\_A = NB\_A.
10. The global matrix *A* must be aligned on a block row boundary; that is, *ia*−1 must be a multiple of MB\_A.
11. The block row offset of *A* must be equal to the block column offset of *A*; that is, mod(*ia*−1, MB\_A) = mod(*ja*−1, NB\_A).

## Error Conditions

### Computational Errors

Matrix *A* is not a positive definite matrix. For details, see the description of the *info* argument.

### Resource Errors

Unable to allocate work space

## Input-Argument and Miscellaneous Errors

### Stage 1:

1. DTYPE\_A is invalid.

### Stage 2:

1. CTEXT\_A is invalid.

### Stage 3:

1. This subroutine was called from outside the process grid.

### Stage 4:

1. *uplo*  $\neq$  'U' or 'L'
2.  $n < 0$
3.  $M\_A < 0$  and  $n = 0$ ;  $M\_A < 1$  otherwise
4.  $N\_A < 0$  and  $n = 0$ ;  $N\_A < 1$  otherwise
5.  $ia < 1$
6.  $ja < 1$
7.  $MB\_A < 1$
8.  $NB\_A < 1$
9.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
10.  $CSRC\_A < 0$  or  $CSRC\_A \geq q$

### Stage 5: If $n \neq 0$ :

1.  $ia > M\_A$
2.  $ja > N\_A$
3.  $ia+n-1 > M\_A$
4.  $ja+n-1 > N\_A$

In all cases:

5.  $MB\_A \neq NB\_A$
6.  $\text{mod}(ia-1, MB\_A) \neq \text{mod}(ja-1, NB\_A)$
7.  $\text{mod}(ia-1, MB\_A) \neq 0$

### Stage 6:

1.  $LLD\_A < \max(1, \text{LOCp}(M\_A))$

### Stage 7:

Each of the following global input arguments are checked to determine whether its value differs from the value specified on process  $P_{00}$ :

1. *uplo* differs.
2.  $n$  differs.
3. *ia* differs.
4. *ja* differs.
5. DTYPE\_A differs.
6. M\_A differs.
7. N\_A differs.
8. MB\_A differs.
9. NB\_A differs.
10. RSRC\_A differs.
11. CSRC\_A differs.

## Examples

### Example 1

This example computes the inverse of a positive definite real symmetric matrix using the  $LL^T$  factorization computed by PDPOTRE. The input transformed matrix  $A$  is the output from PDPOTRE, “Example 1” on page 453.

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)
```

```
          UPLO  N   A   IA  JA  DESC_A  INFO
          |    |   |   |   |         |
CALL PDPOTRI( 'L', 9 , A , 1 , 1 , DESC_A , INFO )
```

	DESC_A
DTYPE_A	1
CTXT_	<i>icontxt</i> <sup>1</sup>
M_	9
N_	9
MB_	3
NB_	3
RSRC_	0
CSRC_	0
LLD_A	See below <sup>2</sup>

#### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:  
 $LLD\_A = \text{MAX}(1, \text{NUMROC}(M\_A, MB\_A, MYROW, RSRC\_A, NPROW))$   
In this example,  $LLD\_A = 6$  on  $P_{00}$  and  $P_{01}$ , and  $LLD\_A = 3$  on  $P_{10}$  and  $P_{11}$ .

#### Output:

Global  $9 \times 9$  inverted positive definite real symmetric matrix  $A$  with block size  $3 \times 3$ :

$$\begin{array}{c}
 \text{B,D} \quad \quad \quad 0 \quad \quad \quad 1 \quad \quad \quad 2 \\
 \begin{array}{c}
 0 \\
 1 \\
 2
 \end{array}
 \left[ \begin{array}{ccc|ccc|ccc}
 2.0 & . & . & . & . & . & . & . & . \\
 -1.0 & 2.0 & . & . & . & . & . & . & . \\
 0.0 & -1.0 & 2.0 & . & . & . & . & . & . \\
 \hline
 0.0 & 0.0 & -1.0 & 2.0 & . & . & . & . & . \\
 0.0 & 0.0 & 0.0 & -1.0 & 2.0 & . & . & . & . \\
 0.0 & 0.0 & 0.0 & 0.0 & -1.0 & 2.0 & . & . & . \\
 \hline
 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -1.0 & 2.0 & . & . \\
 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -1.0 & 2.0 & . \\
 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -1.0 & 1.0
 \end{array} \right]
 \end{array}$$

## PDPOTRI and PZPOTRI

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

Local arrays for A:

p,q	0	1
0	2.0 . . . . -1.0 2.0 . . . . 0.0 -1.0 2.0 . . . . 0.0 0.0 0.0 2.0 . . . . 0.0 0.0 0.0 -1.0 2.0 . . . . 0.0 0.0 0.0 0.0 -1.0 1.0 . . . .	. . . . . . . . . . . . 0.0 0.0 -1.0 . . . . 0.0 0.0 0.0 . . . . 0.0 0.0 0.0 . . . .
1	0.0 0.0 -1.0 . . . . 0.0 0.0 0.0 . . . . 0.0 0.0 0.0 . . . .	2.0 . . . . -1.0 2.0 . . . . 0.0 -1.0 2.0 . . . .

The value of *info* is 0 on all processes.

### Example 2

This example computes the inverse of a positive definite complex Hermitian matrix using the  $LL^H$  factorization computed by PZPOTRF. The input transformed matrix A is the output from PZPOTRF, "Example 2" on page 455.

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      UPLO N  A  IA  JA  DESC_A  INFO
      |  |  |  |  |  |  |
CALL PZPOTRI( 'L', 9 , A , 1 , 1 , DESC_A , INFO )
```

	DESC_A
DTYPE_A	1
CTXT_	<i>icontxt</i> <sup>1</sup>
M_	9
N_	9
MB_	3
NB_	3
RSRC_	0
CSRC_	0
LLD_A	See below <sup>2</sup>

#### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:

LLD\_A = MAX(1, NUMROC(M\_A, MB\_A, MYROW, RSRC\_A, NPROW))

In this example, LLD\_A = 6 on P<sub>00</sub> and P<sub>01</sub>, and LLD\_A = 3 on P<sub>10</sub> and P<sub>11</sub>.

### Output:

Global  $9 \times 9$  inverted positive definite complex Hermitian matrix  $A$  with block size  $3 \times 3$ :

B,D	0	1	2
0	$\begin{pmatrix} (0.06, 0.00) & . & . \\ (0.00, 0.00) & (0.06, 0.00) & . \\ (0.00, 0.00) & (-0.01, 0.00) & (0.07, 0.00) \end{pmatrix}$	$\begin{pmatrix} . & . & . \\ . & . & . \\ . & . & . \end{pmatrix}$	$\begin{pmatrix} . & . & . \\ . & . & . \\ . & . & . \end{pmatrix}$
1	$\begin{pmatrix} (0.00, 0.00) & (0.00, 0.00) & (-0.01, 0.00) \\ (0.00, 0.00) & (0.00, 0.00) & (-0.01, 0.00) \\ (0.00, 0.00) & (0.00, 0.00) & (-0.01, 0.00) \end{pmatrix}$	$\begin{pmatrix} (0.07, 0.00) & . & . \\ (-0.01, -0.01) & (0.09, 0.00) & . \\ (-0.01, 0.00) & (-0.02, -0.01) & (0.11, 0.00) \end{pmatrix}$	$\begin{pmatrix} . & . & . \\ . & . & . \\ . & . & . \end{pmatrix}$
2	$\begin{pmatrix} (0.00, 0.00) & (0.00, 0.00) & (0.00, 0.00) \\ (0.00, 0.00) & (0.00, 0.00) & (0.00, 0.00) \\ (0.00, 0.00) & (0.00, 0.00) & (0.00, 0.00) \end{pmatrix}$	$\begin{pmatrix} (-0.01, 0.00) & (-0.01, 0.00) & (-0.02, -0.01) \\ (-0.01, 0.00) & (-0.01, 0.00) & (-0.02, -0.01) \\ (0.00, 0.00) & (-0.01, 0.00) & (-0.02, 0.01) \end{pmatrix}$	$\begin{pmatrix} (0.15, 0.00) & . & . \\ (-0.04, -0.02) & (0.22, 0.00) & . \\ (-0.05, 0.01) & (-0.13, -0.03) & (0.22, 0.00) \end{pmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for  $A$ :

p,q	0	1
0	$\begin{pmatrix} (0.06, 0.00) & . & . \\ (0.00, 0.00) & (0.06, 0.00) & . \\ (0.00, 0.00) & (-0.01, 0.00) & (0.07, 0.00) \\ (0.00, 0.00) & (0.00, 0.00) & (0.00, 0.00) & (0.15, 0.00) \\ (0.00, 0.00) & (0.00, 0.00) & (0.00, 0.00) & (-0.04, -0.02) & (0.22, 0.00) \\ (0.00, 0.00) & (0.00, 0.00) & (0.00, 0.00) & (-0.05, 0.01) & (-0.13, -0.03) & (0.22, 0.00) \end{pmatrix}$	$\begin{pmatrix} . & . & . \\ . & . & . \\ . & . & . \\ (-0.01, 0.00) & (-0.01, 0.00) & (-0.02, -0.01) \\ (-0.01, 0.00) & (-0.01, 0.00) & (-0.02, -0.01) \\ (0.00, 0.00) & (-0.01, 0.00) & (-0.02, 0.01) \end{pmatrix}$
1	$\begin{pmatrix} (0.00, 0.00) & (0.00, 0.00) & (-0.01, 0.00) & . & . & . \\ (0.00, 0.00) & (0.00, 0.00) & (-0.01, 0.00) & . & . & . \\ (0.00, 0.00) & (0.00, 0.00) & (-0.01, 0.00) & . & . & . \end{pmatrix}$	$\begin{pmatrix} (0.07, 0.00) & . & . \\ (-0.01, -0.01) & (0.09, 0.00) & . \\ (-0.01, 0.00) & (-0.02, -0.01) & (0.11, 0.00) \end{pmatrix}$

The value of *info* is 0 on all processes.

## PDPOCON and PZPOCON — Estimation of the Reciprocal of the Condition Number of a Positive Definite Real Symmetric or Complex Hermitian Matrix

### Purpose

PDPOCON and PZPOCON estimate the reciprocal of the condition number of positive definite real symmetric or complex Hermitian matrix *A*. These subroutines use the results of the factorization of matrix *A*, produced by a preceding call to PDPOTRF or PZPOTRF, respectively. For details on the factorization, see “PDPOTRF and PZPOTRF — Positive Definite Real Symmetric or Complex Hermitian Matrix Factorization” on page 449.

If  $n = 0$ , the subroutines return with  $rcond = 1.0$

See references [17], [19], [23], [38], and [39].

Table 81. Data Types

<i>A, work</i>	<i>anorm, rcond</i>	<i>iwork</i>	<i>rwork</i>	Subroutine
Long-precision real	Long-precision real	Integer		PDPOCON
Long-precision complex	Long-precision real		Long-precision real	PZPOCON

### Syntax

<b>Fortran</b>	CALL PDPOCON ( <i>uplo, n, a, ia, ja, desc_a, anorm, rcond, work, lwork, iwork, liwork, info</i> ) CALL PZPOCON ( <i>uplo, n, a, ia, ja, desc_a, anorm, rcond, work, lwork, rwork, lrwork, info</i> )
<b>C and C++</b>	pdpocon ( <i>uplo, n, a, ia, ja, desc_a, anorm, rcond, work, lwork, iwork, liwork, info</i> ); pzpocon ( <i>uplo, n, a, ia, ja, desc_a, anorm, rcond, work, lwork, rwork, lrwork, info</i> );

### On Entry

*uplo* indicates whether the upper or lower triangular part of the global submatrix *A* is referenced, where:

If *uplo* = 'U', the upper triangular part is referenced.

If *uplo* = 'L', the lower triangular part is referenced.

Scope: **global**

Specified as: a single character; *uplo* = 'U' or 'L'.

*n* is the order of the factored submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

*a* is the local part of the global matrix *A*, containing the factorization of matrix *A* produced by a preceding call to PDPOTRF or PZPOTRF. This identifies the **first element** of the local array *A*. This subroutine computes the location of the first element of the local subarray used, based on *ia, ja, desc\_a, p, q, myrow*, and *mycol*; therefore, the leading LOCp(*ia+n-1*) by LOCq(*ja+n-1*) part of the local array *A* must contain the local pieces of the leading *ia+n-1* by *ja+n-1* part of the global matrix, and:

- If  $uplo = 'U'$ , the leading  $n \times n$  upper triangular part of the global submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  must contain the upper triangular part of the submatrix, and the strictly lower triangular part is not referenced.
- If  $uplo = 'L'$ , the leading  $n \times n$  lower triangular part of the global submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  must contain the lower triangular part of the submatrix, and the strictly upper triangular part is not referenced.

Scope: **local**

Specified as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 81 on page 476. Details about the square block-cyclic data distribution of global matrix  $A$  are stored in  $desc\_a$ .

$ia$  is the row index of the global matrix  $A$ , identifying the first row of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq M\_A$  and  $ia+n-1 \leq M\_A$ .

$ja$  is the column index of the global matrix  $A$ , identifying the first column of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$  and  $ja+n-1 \leq N\_A$ .

$desc\_a$  is the array descriptor for global matrix  $A$ , described in the following table:

$desc\_a$	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_A < p$	Global
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < q$	Global
9	LLD_A	The leading dimension of the local array	$LLD\_A \geq \max(1, LOCp(M\_A))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

$anorm$  is the one norm of the original matrix.

**Note:** You may obtain the value of *anorm* by a preceding call to PDLANSY or PZLANHE. Refer to “PDLANSY, PZLANSY, and PZLANHE — Real Symmetric, Complex Symmetric, or Complex Hermitian Matrix Norm” on page 879.

Scope: **global**

Specified as: long-precision real number  $\geq 0.0$

*rcond* See On Return.

*work* has the following meaning:

If *lwork* = 0, *work* is ignored.

If *lwork*  $\neq$  0, *work* is a work area used by this subroutine, where:

- If *lwork*  $\neq$  -1, then its size is (at least) of length *lwork*.
- If *lwork* = -1, then its size is (at least) of length 1.

Scope: **local**

Specified as: an area of storage containing numbers of data type indicated in Table 81 on page 476.

*lwork* is the number of elements in array WORK.

Scope:

- If *lwork*  $\geq$  0, *lwork* is **local**.
- If *lwork* = -1, *lwork* is **global**.

Specified as: a fullword integer; where:

- If *lwork* = 0, PDPOCON and PZPOCON dynamically allocate the work area used by the subroutine. The work area is deallocated before control is returned to the calling program. This option is an extension to the ScaLAPACK standard.
- If *lwork* = -1, PDPOCON and PZPOCON perform a work area query and return the minimum size of *work* in *work*<sub>1</sub>. No computation is performed and the subroutine returns after error checking is complete.
- Otherwise, use the following rules to determine the value to specify:  
For PDPOCON,  $lwork \geq 2np0 + 2nq0 + \max(2, \max(nb(\max(1, \text{iceil}(nprow-1, npcol))), nq0 + nb(\max(1, \text{iceil}(npcol-1, nprow))))$   
For PZPOCON,  $lwork \geq 2np0 + \max(2, \max(nb(\max(1, \text{iceil}(nprow-1, npcol))), nq0 + nb(\max(1, \text{iceil}(npcol-1, nprow))))$

where:

- *mb* = MB\_A
- *nb* = NB\_A
- *iroff* = mod(*ia*-1, *mb*)
- *icoff* = mod(*ja*-1, *nb*)
- *iarow* = mod(RSRC\_A + (*ia*-1)/*mb*, *nprow*)
- *iacol* = mod(CSRC\_A + (*ja*-1)/*nb*, *npcol*)
- *np0* = NUMROC(*n*+*iroff*, *mb*, *myrow*, *iarow*, *nprow*)
- *nq0* = NUMROC(*n*+*icoff*, *nb*, *mycol*, *iacol*, *npcol*)

*iwork* has the following meaning:

If *liwork* = 0, *iwork* is ignored.

If *liwork*  $\neq$  0, *iwork* is a work area used by this subroutine, where:

- If *liwork*  $\neq$  -1, then its size is (at least) of length *liwork*.
- If *liwork* = -1, then its size is (at least) of length 1.



Scope: **local**

Specified as: an area of storage containing fullword integers.

*liwork* is the number of elements in array *iwork*.

Scope:

- If *liwork*  $\geq 0$ , *liwork* is **local**.
- If *liwork* = -1, *liwork* is **global**.

Specified as: a fullword integer; where:

- If *liwork* = 0, PDPOCON dynamically allocates the work area used by the subroutine. The work area is deallocated before control is returned to the calling program. This option is an extension to the ScaLAPACK standard.
- If *liwork* = -1, PDPOCON performs a work area query and returns the minimum size of *iwork* in *iwork*<sub>1</sub>. No computation is performed and the subroutine returns after error checking is complete.
- Otherwise, use the following rules to determine the value to specify:

$$liwork \geq np0$$

where:

- *mb* = MB\_A
- *irow* = mod(*ia*-1, *mb*)
- *iarow* = mod(RSRC\_A + (*ia*-1)/*mb*, *nprow*)
- *np0* = NUMROC(*n*+*irow*, *mb*, *myrow*, *iarow*, *nprow*)

*rwork* has the following meaning:

If *lrwork* = 0, *rwork* is ignored.

If *lrwork*  $\neq 0$ , *rwork* is a work area used by this subroutine, where:

- If *lrwork*  $\neq -1$ , then its size is (at least) of length *lrwork*.
- If *lrwork* = -1, then its size is (at least) of length 1.

Scope: **local**

Specified as: an area of storage containing long-precision real numbers.

*lrwork* is the number of elements in array *rwork*.

Scope:

- If *lrwork*  $\geq 0$ , *lrwork* is **local**.
- If *lrwork* = -1, *lrwork* is **global**.

Specified as: a fullword integer; where:

- If *lrwork* = 0, PZPOCON dynamically allocates the work area used by the subroutine. The work area is deallocated before control is returned to the calling program. This option is an extension to the ScaLAPACK standard.
- If *lrwork* = -1, PZPOCON performs a work area query and returns the minimum size of *rwork* in *rwork*<sub>1</sub>. No computation is performed and the subroutine returns after error checking is complete.
- Otherwise, use the following rules to determine the value to specify:

$$lrwork \geq 2np0$$

where:

- *nb* = NB\_A
- *icoff* = mod(*ja*-1, *nb*)
- *iacol* = mod(CSRC\_A + (*ja*-1)/*nb*, *npcol*)

–  $nq0 = \text{NUMROC}(n+icoff, nb, mycol, iacol, npcol)$

*info* See On Return.

### On Return

*rcond* has the following meaning:

If  $info = 0$ , an estimate of the reciprocal of the condition number of matrix  $A$  is returned.

If  $n = 0$ , the subroutines return with  $rcond = 1.0$

Else if  $n \neq 0$  and  $anorm = 0.0$ , the subroutines return with  $rcond = 0.0$

Scope: **global**

Returned as: a long-precision real number;  $rcond \geq 0.0$ .

*info* has the following meaning:

If  $info = 0$ , the computation completed normally.

Scope: **global**

Returned as: a fullword integer;  $info = 0$ .

## Notes and Coding Rules

1. In your C program, arguments *rcond* and *info* must be passed by reference.
2. This subroutine accepts lowercase letters for the *uplo* argument.
3.  $A$ , *work*, *iwork*, and *rwork* must have no common elements; otherwise, results are unpredictable.
4. The scalar data specified for input argument  $n$  must be the same for PDLANSY/PZLANHE, PDPOTRF/PZPOTRF, and PDPOCON/PZPOCON.
5. The global submatrix for  $A$  input to PDLANSY/PZLANHE must be the same as the corresponding input arguments for PDPOTRF/PZPOTRF; and thus, the scalar data specified for *ia*, *ja*, and the contents of *desc\_a* must also be the same.
6. The global submatrix for  $A$  input to PDPOCON/PZPOCON must be the same as the corresponding output arguments for PDPOTRF/PZPOTRF; and thus, the scalar data specified for *ia*, *ja*, and the contents of *desc\_a* must also be the same.
7. The NUMROC utility subroutine can be used to determine the values of LOCp(M\_) and LOCq(N\_) used in the argument descriptions above. For details, see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28 and “NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process” on page 868.
8. For suggested block sizes, see “Coding Tips for Optimizing Parallel Performance” on page 77.
9. On both input and output, matrix  $A$  conforms to ScaLAPACK format.
10. The global matrix  $A$  must be distributed using a square block-cyclic distribution; that is,  $MB\_A = NB\_A$ .
11. The global matrix  $A$  must be aligned on a block row boundary; that is,  $ia-1$  must be a multiple of  $MB\_A$ .
12. The block row offset of  $A$  must be equal to the block column offset of  $A$ ; that is,  $\text{mod}(ia-1, MB\_A) = \text{mod}(ja-1, NB\_A)$ .

## Error Conditions

### Computational Errors

None.

### Resource Errors

Unable to allocate work space.

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. DTYPE\_A is invalid.

#### Stage 2:

1. CTXT\_A is invalid.

#### Stage 3:

1. This subroutine was called from outside the process grid.

#### Stage 4:

1.  $uplo \neq 'U' \text{ or } 'L'$
2.  $n < 0$
3.  $M\_A < 1$  and  $n > 0$
4.  $N\_A < 1$  and  $n > 0$
5.  $M\_A < 0$  and  $n = 0$ ;  $M\_A < 1$  otherwise
6.  $N\_A < 0$  and  $n = 0$ ;  $N\_A < 1$  otherwise
7.  $ia < 1$
8.  $ja < 1$
9.  $MB\_A < 1$
10.  $NB\_A < 1$
11.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
12.  $CSRC\_A < 0$  or  $CSRC\_A \geq q$
13.  $anorm < 0$

#### Stage 5: If $n \neq 0$ :

1.  $ia > M\_A$
2.  $ja > N\_A$
3.  $ia+n-1 > M\_A$
4.  $ja+n-1 > N\_A$

In all cases:

5.  $MB\_A \neq NB\_A$
6.  $\text{mod}(ia-1, MB\_A) \neq \text{mod}(ja-1, NB\_A)$
7.  $\text{mod}(ia-1, MB\_A) \neq 0$

#### Stage 6:

1.  $LLD\_A < \max(1, \text{LOCp}(M\_A))$
2.  $lwork \neq 0$ ,  $lwork \neq -1$ , and  $lwork < (\text{minimum value})$ . (For the minimum value, see the *lwork* argument description.)
3.  $liwork \neq 0$ ,  $liwork \neq -1$ , and  $liwork < (\text{minimum value})$ . (For the minimum value, see the *liwork* argument description.)
4.  $lrwork \neq 0$ ,  $lrwork \neq -1$ , and  $lrwork < (\text{minimum value})$ . (For the minimum value, see the *lrwork* argument description.)

#### Stage 7:

## PDPOCON and PZPOCON

Each of the following global input arguments are checked to determine whether its value differs from the value specified on process  $P_{00}$ :

1. *uplo* differs.
2. *n* differs.
3. *ia* differs.
4. *ja* differs.
5. *DTYPE\_A* differs.
6. *M\_A* differs.
7. *N\_A* differs.
8. *MB\_A* differs.
9. *NB\_A* differs.
10. *RSRC\_A* differs.
11. *CSRC\_A* differs.
12. *anorm* differs.

Also:

13. *lwork* = -1 on a subset of processes.
14. *liwork* = -1 on a subset of processes.
15. *lrwork* = -1 on a subset of processes.

## Examples

### Example 1

This example estimates the reciprocal of the condition number of positive definite real symmetric matrix *A*. The input matrix *A* to PDLANSY and PDPOTRF in this example is the same as input matrix *A* in the PDPOTRF “Example 1” on page 453.

#### Notes:

1. Because *WORK* is used by both PDLANSY and PDPOCON, we do not dynamically allocate the *WORK* area.
2. Because *liwork* = 0, PDPOCON dynamically allocates the *IWORK* area used by this subroutine.

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      NORM UPLO  N   A   IA   JA   DESC_A  WORK
      |    |    |   |   |   |   |
ANORM = PDLANSY( 'L', 'L', 9 , A , 1 , 1, DESC_A, WORK )

      UPLO  N   A   IA   JA   DESC_A  INFO
      |    |   |   |   |   |
CALL PDPOTRF( 'L', 9 , A , 1 , 1 , DESC_A , INFO )

      UPLO  N   A   IA   JA   DESC_A  ANORM  RCOND  WORK  LWORK  IWORK  LIWORK  INFO
      |    |   |   |   |   |   |   |   |   |   |   |
CALL PDPOCON( 'L', 9 , A , 1 , 1 , DESC_A , ANORM , RCOND , WORK , 100 , IWORK , 0 , INFO )
```

	DESC_A
DTYPE_A	1
CTXT_	<i>icontxt</i> <sup>1</sup>

	DESC_A
M_	9
N_	9
MB_	3
NB_	3
RSRC_	0
CSRC_	0
LLD_A	See below <sup>2</sup>

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_A as follows:  

$$\text{LLD\_A} = \text{MAX}(1, \text{NUMROC}(\text{M\_A}, \text{MB\_A}, \text{MYROW}, \text{RSRC\_A}, \text{NPROW}))$$
In this example,  $\text{LLD\_A} = 6$  on  $P_{00}$  and  $P_{01}$ , and  $\text{LLD\_A} = 3$  on  $P_{10}$  and  $P_{11}$ .

**Output:**

The value of *rcond* = 0.5556D-02 on all processes.

The value of *info* is 0 on all processes.

**Example 2**

This example estimates the reciprocal of the condition number of positive definite complex Hermitian matrix *A*. The input matrix *A* to PZLANHE and PZPOTRF is the same as input matrix *A* in the PZPOTRF “Example 2” on page 455.

**Notes:**

1. Because RWORK is used by both PZLANHE and PZPOCON, we do not dynamically allocate the RWORK area.
2. Because *lwork* = 0, PZPOCON dynamically allocates the WORK area used by this subroutine.

**Call Statements and Input:**

```

ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      NORM UPLO  N   N   A  IA  JA  DESC_A  WORK
      |    |    |   |   |  |  |    |    |
ANORM = PZLANHE( 'L', 'L', 9 , 9,  A , 1 , 1, DESC_A, RWORK )

      UPLO  N   A  IA  JA  DESC_A  INFO
      |    |   |  |  |  |    |    |
CALL PZPOTRF( 'L', 9 , A , 1 , 1, DESC_A, INFO )

      UPLO  N   A  IA  JA  DESC_A  ANORM  RCOND  WORK  LWORK  RWORK  LRWORK  INFO
      |    |   |  |  |  |    |    |    |    |    |    |    |
CALL PZPOCON( 'L', 9,  A, 1, 1, DESC_A, ANORM, RCOND, WORK, 0,  RWORK, 100, INFO )

```

## PDPOCON and PZPOCON

	DESC_A
DTYPE_A	1
CTXT_	<i>icontxt</i> <sup>2</sup>
M_	9
N_	9
MB_	3
NB_	3
RSRC_	0
CSRC_	0
LLD_A	See below <sup>2</sup>

### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_A as follows:  

$$\text{LLD\_A} = \text{MAX}(1, \text{NUMROC}(\text{M\_A}, \text{MB\_A}, \text{MYROW}, \text{RSRC\_A}, \text{NPROW}))$$
In this example, LLD\_A = 6 on P<sub>00</sub> and P<sub>01</sub>, and LLD\_A = 3 on P<sub>10</sub> and P<sub>11</sub>.

### Output:

The value of *rcond* = 0.2773D-01 on all processes.

The value of *info* is 0 on all processes.

---

## Banded Linear Algebraic Equation Subroutines

This section contains the banded linear algebraic equation subroutine descriptions.

## PDPBSV — Positive Definite Symmetric Band Matrix Factorization and Solve

### Purpose

This subroutine solves the following system of equations for multiple right-hand sides:

- $AX = B$

where, in the formula above:

- $A$  represents the global positive definite symmetric band submatrix  $A_{ja:ja+n-1, ja:ja+n-1}$  to be factored by Cholesky factorization.
- $B$  represents the global general submatrix  $B_{ib:ib+n-1, 1:nrhs}$  containing the right-hand sides in its columns.
- $X$  represents the global general submatrix  $B_{ib:ib+n-1, 1:nrhs}$  containing the output solution vectors in its columns.

If  $n$  is 0, no computation is performed and the subroutine returns after doing some parameter checking. If  $n > 0$  and  $nrhs$  is 0, no solutions are computed and the subroutine returns after factoring the matrix.

See references [2], [24], [41], and [42].

Table 82. Data Types

$A, B, work$	Subroutine
Long-precision real	PDPBSV

### Syntax

Fortran	CALL PDPBSV ( <i>uplo</i> , <i>n</i> , <i>k</i> , <i>nrhs</i> , <i>a</i> , <i>ja</i> , <i>desc_a</i> , <i>b</i> , <i>ib</i> , <i>desc_b</i> , <i>work</i> , <i>lwork</i> , <i>info</i> )
C and C++	pdpbsv ( <i>uplo</i> , <i>n</i> , <i>k</i> , <i>nrhs</i> , <i>a</i> , <i>ja</i> , <i>desc_a</i> , <i>b</i> , <i>ib</i> , <i>desc_b</i> , <i>work</i> , <i>lwork</i> , <i>info</i> );

### On Entry

*uplo* indicates whether the upper or lower triangular part of the global submatrix  $A$  is referenced, where:

If *uplo* = 'U', the upper triangular part is referenced.

If *uplo* = 'L', the lower triangular part is referenced.

Scope: **global**

Specified as: a single character; *uplo* = 'U' or 'L'.

*n* is the number of columns in the submatrix  $A$ , stored in the upper- or lower-band-packed storage mode. It is also the number of rows in the general submatrix  $B$  containing the multiple right-hand sides.

Scope: **global**

Specified as: a fullword integer;  $0 \leq n \leq (NB\_A)p - \text{mod}(ja-1, NB\_A)$ .

*k* is the half bandwidth of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer, where:



- If  $uplo = 'U'$ ,  $0 \leq k \leq NB\_A$ .
- If  $uplo = 'L'$ ,  $0 \leq k < n$ .

These limits for  $k$  are extensions of the ScaLAPACK standard.

$nrhs$  is the number of columns in submatrix  $B$  used in the computation.

Scope: **global**

Specified as: a fullword integer;  $nrhs \geq 0$ .

- $a$  is the local part of the global positive definite symmetric band matrix  $A$ , stored in upper- or lower-band-packed storage mode, to be factored. This identifies the **first element** of the local array  $A$ . This subroutine computes the location of the first element of the local subarray used, based on  $k$ ,  $ja$ ,  $desc\_a$ , and  $p$ ; therefore, the leading  $k+1$  by  $LOCp(ja+n-1)$  part of the local array  $A$  must contain the local pieces of the leading  $k+1$  by  $ja+n-1$  part of the global matrix, and:
- If  $uplo = 'U'$ , the leading  $n \times n$  upper triangular part of the global submatrix  $A_{ja:ja+n-1, ja:ja+n-1}$  must contain the upper triangular part of the submatrix, and the strictly lower triangular part is not referenced.
  - If  $uplo = 'L'$ , the leading  $n \times n$  lower triangular part of the global submatrix  $A_{ja:ja+n-1, ja:ja+n-1}$  must contain the lower triangular part of the submatrix, and the strictly upper triangular part is not referenced.

Scope: **local**

Specified as: an  $LLD\_A$  by (at least)  $LOCp(ja+n-1)$  array, containing numbers of the data type indicated in Table 82 on page 486. Details about the block-cyclic data distribution of global matrix  $A$  are stored in  $desc\_a$ .

On output, array  $A$  is overwritten; that is, original input is not preserved.

$ja$  is the column index of the global matrix  $A$ , identifying the first column of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$  and  $ja+n-1 \leq N\_A$ .

$desc\_a$  is the array descriptor for global matrix  $A$ , which may be type 501 or type 1, as described in the following tables. For rules on using array descriptors, see "Notes and Coding Rules" on page 491.

$desc\_a$	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor Type	DTYPE_A = 501 for $1 \times p$ or $p \times 1$  where $p$ is the number of processes in a process grid.	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	N_A	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
4	NB_A	Column block size	$NB\_A \geq 1$ and $0 \leq n \leq (NB\_A)p - \text{mod}(ja-1, NB\_A)$	Global

<i>desc_a</i>	Name	Description	Limits	Scope
5	CSRC_A	The process column over which the first column of the global matrix is distributed	$0 \leq \text{CSRC\_A} < p$	Global
6	LLD_A	Leading dimension	$\text{LLD\_A} \geq k+1$	Local
7	—	Reserved	—	—

Specified as: an array of (at least) length 7, containing fullword integers.

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	$\text{DTYPE\_A} = 1$ for $1 \times p$ where $p$ is the number of processes in a process grid.	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	$M\_A > k$	Global
4	N_A	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$\text{MB\_A} \geq 1$	Global
6	NB_A	Column block size	$\text{NB\_A} \geq 1$ and $0 \leq n \leq (\text{NB\_A})p - \text{mod}(ja-1, \text{NB\_A})$	Global
7	RSRC_A	The process row over which the first row of the global matrix is distributed	$\text{RSRC\_A} = 0$	Global
8	CSRC_A	The process column over which the first column of the global matrix is distributed	$0 \leq \text{CSRC\_A} < p$	Global
9	LLD_A	The leading dimension of the local array	$\text{LLD\_A} \geq k+1$	Local

Specified as: an array of (at least) length 9, containing fullword integers.

- b* is the local part of the global general matrix **B**, containing the multiple right-hand sides of the system. This identifies the **first element** of the local array **B**. This subroutine computes the location of the first element of the local subarray used, based on *ib*, *desc\_b*, and *p*; therefore, the leading  $\text{LOCp}(ib+n-1)$  by *nrhs* part of the local array **B** must contain the local pieces of the leading  $ib+n-1$  by *nrhs* part of the global matrix.

Scope: **local**

Specified as: an LLD\_B by (at least) *nrhs* array, containing numbers of the data type indicated in Table 82 on page 486. Details about the block-cyclic data distribution of global matrix **B** are stored in *desc\_b*.

- ib* is the row index of the global matrix **B**, identifying the first row of the submatrix **B**.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ib \leq M\_B$  and  $ib+n-1 \leq M\_B$ .

*desc\_b* is the array descriptor for global matrix *B*, which may be type 502 or type 1, as described in the following tables. For rules on using array descriptors, see “Notes and Coding Rules” on page 491.

<i>desc_b</i>	Name	Description	Limits	Scope
1	DTYPE_B	Descriptor type	DTYPE_B = 502 for $p \times 1$ or $1 \times p$ where $p$ is the number of processes in a process grid.	Global
2	CTXT_B	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_B	Number of rows in the global matrix	If $n = 0$ : M_B $\geq 0$ Otherwise: M_B $\geq 1$	Global
4	MB_B	Row block size	MB_B $\geq 1$ and $0 \leq n \leq (MB\_B)p - \text{mod}(ib-1, MB\_B)$	Global
5	RSRC_B	The process row over which the first row of the global matrix is distributed	$0 \leq \text{RSRC\_B} < p$	Global
6	LLD_B	Leading dimension	LLD_B $\geq \max(1, \text{LOCp}(M\_B))$	Local
7	—	Reserved	—	—

Specified as: an array of (at least) length 7, containing fullword integers.

<i>desc_b</i>	Name	Description	Limits	Scope
1	DTYPE_B	Descriptor type	DTYPE_B = 1 for $p \times 1$ where $p$ is the number of processes in a process grid.	Global
2	CTXT_B	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_B	Number of rows in the global matrix	If $n = 0$ : M_B $\geq 0$ Otherwise: M_B $\geq 1$	Global
4	N_B	Number of columns in the global matrix	N_B $\geq nrhs$	Global
5	MB_B	Row block size	MB_B $\geq 1$ and $0 \leq n \leq (MB\_B)p - \text{mod}(ib-1, MB\_B)$	Global
6	NB_B	Column block size	NB_B $\geq 1$	Global
7	RSRC_B	The process row over which the first row of the global matrix is distributed	$0 \leq \text{RSRC\_B} < p$	Global
8	CSRC_B	The process column over which the first column of the global matrix is distributed	CSRC_B=0	Global
9	LLD_B	Leading dimension	LLD_B $\geq \max(1, \text{LOCp}(M\_B))$	Local

Specified as: an array of (at least) length 9, containing fullword integers.

*work* has the following meaning:

If  $lwork = 0$ , *work* is ignored.

If  $lwork \neq 0$ , *work* is the work area used by this subroutine, where:

- If  $lwork \neq -1$ , the size of *work* is (at least) of length *lwork*.
- If  $lwork = -1$ , the size of *work* is (at least) of length 1.

Scope: **local**

Specified as: an area of storage containing numbers of data type indicated in Table 82 on page 486.

*lwork* is the number of elements in array WORK.

Scope:

- If  $lwork \geq 0$ , *lwork* is **local**
- If  $lwork = -1$ , *lwork* is **global**

Specified as: a fullword integer, where:

- If  $lwork = 0$ , PDPBSV dynamically allocates the work area used by this subroutine. The work area is deallocated before control is returned to the calling program. This option is an extension to the ScaLAPACK standard. It is suggested that you specify  $lwork=0$ .
- If  $lwork = -1$ , PDPBSV performs a work area query and returns the optimum size of *work* in  $work_1$ . No computation is performed and the subroutine returns after error checking is complete.
- Otherwise, it must have the following value:  

$$lwork \geq (NB\_A+2k)(k)+\max(nrhs, k)(k)$$

*info* See On Return.

## On Return

*a* *a* is overwritten; that is, the original input is not preserved. This subroutine overwrites data in positions that do not contain the positive definite symmetric band matrix *A* stored in upper- or lower-band-packed storage mode.

*b* *b* is the updated local part of the global matrix *B*, containing the solution vectors.

Scope: **local**

Returned as: an LLD\_B by (at least) *nrhs* array, containing numbers of the data type indicated in Table 82 on page 486.

*work* is the work area used by this subroutine if  $lwork \neq 0$ , where:

If  $lwork \neq 0$  and  $lwork \neq -1$ , the size of *work* is (at least) of length *lwork*.

If  $lwork = -1$ , the size of *work* is (at least) of length 1.

Scope: **local**

Returned as: an area of storage, containing numbers of the data type indicated in Table 82 on page 486, where:

- If  $lwork \geq 1$ ,  $work_1$  is set to the minimum *lwork* value needed.
- If  $lwork = -1$ ,  $work_1$  is set to the optimum *lwork* value needed.

Except for  $work_1$ , the contents of *work* are overwritten on return.

*info* has the following meaning:

If *info* = 0, global submatrix *A* is positive definite, and the factorization completed normally or the work area query completed successfully.

If *info* > 0, the leading minor of order *i* of the global submatrix *A* is not positive definite. *info* is set equal to *i*, where the first leading minor was encountered at  $A_{ja+i-1, ja+i-1}$ . The results contained in matrix *A* are not defined.

Scope: **global**

Returned as: a fullword integer; *info* ≥ 0.

## Notes and Coding Rules

1. In your C program, argument *info* must be passed by reference.
2. If *n* > 0 and *nrhs* = 0, only the factorization is completed.
3. The subroutine accepts lowercase letters for the *uplo* argument.
4. This subroutine gives the best performance for wide band widths, for example:

$$k > 100\sqrt{p}$$

where *p* is the number of processes. For details, see references [2], [41], and [42]. Also, it is suggested that you specify *uplo* = 'L'.

5. *A*, *B*, and *work* must have no common elements; otherwise, results are unpredictable.
6. In all cases, follow these rules:
  - *ib* = *ja*
  - DTYPE\_A=501 or 1
  - DTYPE\_B=502 or 1
  - NB\_A = MB\_B
  - If DTYPE\_A=1, RSRC\_A=0, M\_A ≥ *k*+1, and MB\_A ≥ 1.
  - If DTYPE\_B=1, CSRC\_B=0, N\_B ≥ *nrhs*, and NB\_B ≥ 1.
  - CTXT\_A = CTXT\_B
  - Following are the consistent combinations of array descriptor types and process grids, where *p* is the number of processes in the process grid:

DTYPE_A	DTYPE_B	Process Grid
501	502	$p \times 1$ or $1 \times p$
501	1	$1 \times p$
1	502	$p \times 1$
1	1	$1 \times 1$

7. To determine the values of LOCp(*n*) used in the argument descriptions, see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28 for descriptor type-1 or “Determining the Number of Rows or Columns in Your Local Arrays” on page 32 for descriptor type-501 and type-502.
8. The global band matrix *A* must be positive definite. If *A* is not positive definite, this subroutine uses the *info* argument to provide information about

$A$  and issues an error message. This differs from ScaLAPACK, which only uses the *info* argument to provide information about  $A$ .

9. The global positive definite symmetric band matrix  $A$  must be stored in upper- or lower-band-packed storage mode. See the section on block-cyclically distributing a symmetric matrix in “Matrices” on page 40.

Matrix  $A$  must be distributed over a one-dimensional process grid using block-cyclic data distribution. For more information on using block-cyclic data distribution, see “Specifying Block-Cyclically-Distributed Matrices for the Banded Linear Algebraic Equations” on page 29.

10. Matrix  $B$  must be distributed over a one-dimensional process grid, using block-cyclic data distribution. For more information on block-cyclic data distribution, see “Specifying Block-Cyclically-Distributed Matrices for the Banded Linear Algebraic Equations” on page 29. Also, see the section on distributing the right-hand side matrix in “Matrices” on page 40.
11. If  $lwork = -1$  on any process, it must equal  $-1$  on all processes. That is, if a subset of the processes specifies  $-1$  for the work area size, they must all specify  $-1$ .
12. Although global matrices  $A$  and  $B$  may be block-cyclically distributed on a  $1 \times p$  or  $p \times 1$  process grid, the values of  $n$ ,  $ja$ ,  $ib$ ,  $NB\_A$  and  $MB\_B$ , must be chosen so that each process has at most one full or partial block of each of the global submatrices  $A$  and  $B$ .

## Error Conditions

### Computational Errors

Matrix  $A$  is not positive definite (corresponding computational error messages are issued by both PDPBTRF and PDPBSV). For details, see the description of the *info* argument.

### Resource Errors

$lwork = 0$  and unable to allocate workspace

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1.  $DTYPE\_A$  is invalid.
2.  $DTYPE\_B$  is invalid.

#### Stage 2:

1.  $CTXT\_A$  is invalid.

#### Stage 3:

1. PDPBSV was called from outside the process grid.

#### Stage 4:

1. The process grid is not  $1 \times p$  or  $p \times 1$ .
2.  $uplo \neq 'U'$  or  $'L'$
3.  $n < 0$
4.  $k < 0$
5.  $k+1 > n$
6.  $ja < 1$
7.  $DTYPE\_A = 1$  and:
  - a.  $M\_A < k+1$
  - b.  $MB\_A < 1$
  - c.  $RSRC\_A \neq 0$

- d. The process grid is not  $1 \times p$ .
- 8.  $N\_A < 0$  and ( $n = 0$ );  $N\_A < 1$  otherwise
- 9.  $NB\_A < 1$
- 10.  $n + \text{mod}(ja-1, NB\_A) > (NB\_A)p$
- 11.  $CSRC\_A < 0$  or  $CSRC\_A \geq p$
- 12.  $uplo = 'U'$  and  $k > NB\_A$
- 13.  $nrhs < 0$
- 14.  $ib \neq ja$
- 15.  $ib < 1$
- 16.  $DTYPE\_B = 1$  and:
  - a.  $N\_B < nrhs$
  - b.  $NB\_B < 1$
  - c.  $CSRC\_B \neq 0$
  - d. The process grid is not  $p \times 1$ .
- 17.  $M\_B < 0$  and ( $n = 0$ );  $M\_B < 1$  otherwise
- 18.  $MB\_B < 1$
- 19.  $n + \text{mod}(ib-1, MB\_B) > (MB\_B)p$
- 20.  $MB\_B \neq NB\_A$
- 21.  $RSRC\_B < 0$  or  $RSRC\_B \geq p$
- 22.  $CTXT\_A \neq CTXT\_B$

**Stage 5:** If  $n \neq 0$ :

- 1.  $ja+n-1 > N\_A$
- 2.  $ja > N\_A$
- 3.  $ib > M\_B$
- 4.  $ib+n-1 > M\_B$
- 5.  $LLD\_A < k+1$

**Stage 6:**

- 1.  $LLD\_B < \max(1, \text{LOCp}(M\_B))$
- 2.  $lwork \neq 0$ ,  $lwork \neq -1$ , and  $lwork < (NB\_A+2k)(k) + \max(nrhs, k)(k)$

**Stage 7:**

Each of the following global input arguments are checked to determine whether its value differs from the value specified on process  $P_{00}$ :

- 1.  $uplo$  differs.
- 2.  $n$  differs.
- 3.  $k$  differs.
- 4.  $nrhs$  differs.
- 5.  $ja$  differs.
- 6.  $DTYPE\_A$  differs.
- 7.  $DTYPE\_A$  does not differ and:
  - a.  $N\_A$  differs.
  - b.  $NB\_A$  differs.
  - c.  $CSRC\_A$  differs.
  - d.  $DTYPE\_A = 1$  and:
    - 1)  $M\_A$  differs.
    - 2)  $MB\_A$  differs.
    - 3)  $RSRC\_A$  differs.
- 8.  $ib$  differs.
- 9.  $DTYPE\_B$  differs.
- 10.  $DTYPE\_B$  does not differ and:
  - a.  $M\_B$  differs.
  - b.  $MB\_B$  differs.
  - c.  $RSRC\_B$  differs.

d. `DTYPE_A = 1` and:

- 1) `N_B` differs.
- 2) `NB_B` differs.
- 3) `CSRC_B` differs.

Also:

11. `lwork = -1` on a subset of processes.

## Examples

### Example

This example shows a factorization of the positive definite symmetric band matrix *A* of order 9 with a half bandwidth of 7:

$$\begin{bmatrix} 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 0.0 \\ 1.0 & 2.0 & 2.0 & 2.0 & 2.0 & 2.0 & 2.0 & 2.0 & 1.0 \\ 1.0 & 2.0 & 3.0 & 3.0 & 3.0 & 3.0 & 3.0 & 3.0 & 2.0 \\ 1.0 & 2.0 & 3.0 & 4.0 & 4.0 & 4.0 & 4.0 & 4.0 & 3.0 \\ 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 5.0 & 5.0 & 5.0 & 4.0 \\ 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 6.0 & 6.0 & 6.0 & 5.0 \\ 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 6.0 & 7.0 & 7.0 & 6.0 \\ 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 6.0 & 7.0 & 8.0 & 7.0 \\ 0.0 & 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 6.0 & 7.0 & 8.0 \end{bmatrix}$$

Matrix *A* is stored in lower-band-packed storage mode:

$$\begin{bmatrix} 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 6.0 & 7.0 & 8.0 & 8.0 \\ 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 6.0 & 7.0 & 7.0 & . \\ 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 6.0 & 6.0 & . & . \\ 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 5.0 & . & . & . \\ 1.0 & 2.0 & 3.0 & 4.0 & 4.0 & . & . & . & . \\ 1.0 & 2.0 & 3.0 & 3.0 & . & . & . & . & . \\ 1.0 & 2.0 & 2.0 & . & . & . & . & . & . \\ 1.0 & 1.0 & . & . & . & . & . & . & . \end{bmatrix}$$

where “.” means you do not have to store a value in that position in the local array. However, these storage positions are required and are overwritten during the computation.

### Notes:

1. On output, the submatrix *A* is overwritten; that is, the original input is not preserved.
2. Notice **only one process grid was created**, even though, `DTYPE_A = 501` and `DTYPE_B = 502`.
3. Because `lwork = 0`, PDPBSV dynamically allocates the work area used by this subroutine.



## Call Statements and Input:

```

ORDER = 'R'
NPROW = 1
NPCOL = 3
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      UPLO  N   K  NRHS A   JA  DESC_A  B   IB  DESC_B  WORK LWORK INFO
      |    |   |   |   |   |         |   |   |         |    |    |
CALL PDPBSV( 'L' , 9 , 7 , 3 , A , 1 , DESC_A , B , 1 , DESC_B , WORK , 0 , INFO )

```

	Desc_A
DTYPE_	501
CTXT_	<i>icontxt</i> <sup>1</sup>
N_	9
NB_	3
CSRC_	0
LLD_A	8
Reserved	—

## Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.

	Desc_B
DTYPE_	502
CTXT_	<i>icontxt</i> <sup>1</sup>
M_	9
MB_	3
RSRC_	0
LLD_B	3
Reserved	—

## Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.

Global matrix *A* stored in lower-band-packed storage mode with block size of 3:

$$\begin{array}{c}
 \text{B,D} \quad \quad \quad 0 \quad \quad \quad 1 \quad \quad \quad 2 \\
 \quad \quad \quad \left[ \begin{array}{ccc|ccc|ccc}
 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 6.0 & 7.0 & 8.0 & 8.0 \\
 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 6.0 & 7.0 & 7.0 & . \\
 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 6.0 & 6.0 & . & . \\
 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 5.0 & . & . & . \\
 1.0 & 2.0 & 3.0 & 4.0 & 4.0 & . & . & . & . \\
 1.0 & 2.0 & 3.0 & 3.0 & . & . & . & . & . \\
 1.0 & 2.0 & 2.0 & . & . & . & . & . & . \\
 1.0 & 1.0 & . & . & . & . & . & . & .
 \end{array} \right]
 \end{array}$$

The following is the  $1 \times 3$  process grid:

B,D	0	1	2
0	P <sub>00</sub>	P <sub>01</sub>	P <sub>02</sub>

Local array A with block size of 3:

p,q	0	1	2
0	1.0 2.0 3.0 1.0 2.0 3.0 1.0 2.0 3.0 1.0 2.0 3.0 1.0 2.0 3.0 1.0 2.0 3.0 1.0 2.0 2.0 1.0 1.0 .	4.0 5.0 6.0 4.0 5.0 6.0 4.0 5.0 6.0 4.0 5.0 5.0 4.0 4.0 . 3.0 . . . . . . . .	7.0 8.0 8.0 7.0 7.0 . 6.0 . . . . . . . . . . . . . . . . .

Global matrix  $B$  with block size of 3:

B,D	0
0	8.0 36.0 44.0 16.0 80.0 80.0 23.0 122.0 108.0
1	29.0 161.0 129.0 34.0 196.0 144.0 38.0 226.0 154.0
2	41.0 250.0 160.0 43.0 267.0 163.0 36.0 240.0 120.0

The following is the  $1 \times 3$  process grid:

B,D	0	1	2
0	P <sub>00</sub>	P <sub>01</sub>	P <sub>02</sub>

Local array B with block size of 3:

p,q	0	1	2
0	8.0 36.0 44.0 16.0 80.0 80.0 23.0 122.0 108.0	29.0 161.0 129.0 34.0 196.0 144.0 38.0 226.0 154.0	41.0 250.0 160.0 43.0 267.0 163.0 36.0 240.0 120.0

**Output:**

Global matrix  $B$  with block size of 3:

B,D	0
0	1.0 1.0 9.0 1.0 2.0 8.0 1.0 3.0 7.0
1	1.0 4.0 6.0 1.0 5.0 5.0 1.0 6.0 4.0
2	1.0 7.0 3.0 1.0 8.0 2.0 1.0 9.0 1.0

The following is the  $1 \times 3$  process grid:

B,D	0	1	2
0	$P_{00}$	$P_{01}$	$P_{02}$

Local array B with block size of 3:

p,q	0			1			2		
0	1.0	1.0	9.0	1.0	4.0	6.0	1.0	7.0	3.0
	1.0	2.0	8.0	1.0	5.0	5.0	1.0	8.0	2.0
	1.0	3.0	7.0	1.0	6.0	4.0	1.0	9.0	1.0

The value of *info* is 0 on all processes.

## PDPBTRF — Positive Definite Symmetric Band Matrix Factorization

### Purpose

This subroutine uses Cholesky factorization to factor a positive definite symmetric band matrix  $A$ , stored in upper- or lower-band-packed storage mode, into one of the following forms:

- $A = \mathbf{U}^T \mathbf{U}$  if  $A$  is upper triangular.
- $A = \mathbf{L} \mathbf{L}^T$  if  $A$  is lower triangular.

where, in the formulas above:

- $A$  represents the global positive definite symmetric band submatrix  $A_{ja:ja+n-1, ja:ja+n-1}$  to be factored.
- $\mathbf{U}$  is an upper triangular matrix.
- $\mathbf{L}$  is a lower triangular matrix.

To solve the system of equations with multiple right-hand sides, follow the call to this subroutine with one or more calls to PDPBTRS. The output from this factorization subroutine should be used only as input to PDPBTRS.

If  $n = 0$ , no computation is performed and the subroutine returns after doing some parameter checking. See references [2], [24], [41], and [42].

Table 83. Data Types

$A, af, work$	Subroutine
Long-precision real	PDPBTRF

### Syntax

Fortran	CALL PDPBTRF ( <i>uplo</i> , <i>n</i> , <i>k</i> , <i>a</i> , <i>ja</i> , <i>desc_a</i> , <i>af</i> , <i>laf</i> , <i>work</i> , <i>lwork</i> , <i>info</i> )
C and C++	pdpbtrf ( <i>uplo</i> , <i>n</i> , <i>k</i> , <i>a</i> , <i>ja</i> , <i>desc_a</i> , <i>af</i> , <i>laf</i> , <i>work</i> , <i>lwork</i> , <i>info</i> );

### On Entry

*uplo* indicates whether the upper or lower triangular part of the global submatrix  $A$  is referenced, where:

If *uplo* = 'U', the upper triangular part is referenced.

If *uplo* = 'L', the lower triangular part is referenced.

Scope: **global**

Specified as: a single character; *uplo* = 'U' or 'L'.

*n* is the number of columns in the submatrix  $A$ , stored in upper- or lower-band-packed storage mode, to be factored.

Scope: **global**

Specified as: a fullword integer;  $0 \leq n \leq (\text{NB\_A})p - \text{mod}(ja-1, \text{NB\_A})$ .

*k* is the half bandwidth of the submatrix  $A$  to be factored.

Scope: **global**

Specified as: a fullword integer, where:

- If *uplo* = 'U',  $0 \leq k \leq \text{NB\_A}$ .
- If *uplo* = 'L',  $0 \leq k < n$ .

These limits for  $k$  are extensions of the ScaLAPACK standard.

$a$  is the local part of the global positive definite symmetric band matrix  $A$ , stored in upper- or lower-band-packed storage mode, to be factored. This identifies the **first element** of the local array  $A$ . This subroutine computes the location of the first element of the local subarray used, based on  $k$ ,  $ja$ ,  $desc\_a$ , and  $p$ ; therefore, the leading  $k+1$  by  $LOCp(ja+n-1)$  part of the local array  $A$  must contain the local pieces of the leading  $k+1$  by  $ja+n-1$  part of the global matrix, and:

- If  $uplo = 'U'$ , the leading  $n \times n$  upper triangular part of the global submatrix  $A_{ja:ja+n-1, ja:ja+n-1}$  must contain the upper triangular part of the submatrix, and the strictly lower triangular part is not referenced.
- If  $uplo = 'L'$ , the leading  $n \times n$  lower triangular part of the global submatrix  $A_{ja:ja+n-1, ja:ja+n-1}$  must contain the lower triangular part of the submatrix, and the strictly upper triangular part is not referenced.

Scope: **local**

Specified as: an  $LLD\_A$  by (at least)  $LOCp(ja+n-1)$  array, containing numbers of the data type indicated in Table 83 on page 498. Details about the block-cyclic data distribution of global matrix  $A$  are stored in  $desc\_a$ .

On output, array  $A$  is overwritten; that is, original input is not preserved.

$ja$  is the column index of the global matrix  $A$ , identifying the first column of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$  and  $ja+n-1 \leq N\_A$ .

$desc\_a$  is the array descriptor for global matrix  $A$ , which may be type 501 or type 1, as described in the following tables.

$desc\_a$	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor Type	DTYPE_A = 501 for $1 \times p$ or $p \times 1$  where $p$ is the number of processes in a process grid.	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	N_A	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
4	NB_A	Column block size	$NB\_A \geq 1$ and $0 \leq n \leq (NB\_A)p - \text{mod}(ja-1, NB\_A)$	Global
5	CSRC_A	The process column over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < p$	Global
6	LLD_A	Leading dimension	$LLD\_A \geq k+1$	<b>Local</b>
7	—	Reserved	—	—

Specified as: an array of (at least) length 7, containing fullword integers.

## PDPBTRF

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A = 1 for $1 \times p$ where $p$ is the number of processes in a process grid.	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	$M\_A > k$	Global
4	N_A	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$ and $0 \leq n \leq (NB\_A)p - \text{mod}(ja-1, NB\_A)$	Global
7	RSRC_A	The process row over which the first row of the global matrix is distributed	$RSRC\_A = 0$	Global
8	CSRC_A	The process column over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < p$	Global
9	LLD_A	The leading dimension of the local array	$LLD\_A \geq k+1$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

*af* is a reserved output area and its size is specified by LAF.

Scope: **local**

Specified as: for migration purposes, you should specify a one-dimensional, long-precision array of (at least) length LAF.

*laf* is the number of elements in array AF.

The *laf* argument must be specified; however, this subroutine currently ignores its value. For migration purposes, you should specify *laf* using the formula below.

Scope: **local**

Specified as: a fullword integer,  $laf \geq (NB\_A + 2k)(k)$ .

*work* has the following meaning:

If  $lwork = 0$ , *work* is ignored.

If  $lwork \neq 0$ , *work* is the work area used by this subroutine, where:

- If  $lwork \neq -1$ , the size of *work* is (at least) of length *lwork*.
- If  $lwork = -1$ , the size of *work* is (at least) of length 1.

Scope: **local**

Specified as: an area of storage containing numbers of data type indicated in Table 83 on page 498.

*lwork* is the number of elements in array WORK.

Scope:

- If  $lwork \geq 0$ ,  $lwork$  is **local**
- If  $lwork = -1$ ,  $lwork$  is **global**

Specified as: a fullword integer; where:

- If  $lwork = 0$ , PDPBTRF dynamically allocates the work area used by this subroutine. The work area is deallocated before control is returned to the calling program. This option is an extension to the ScaLAPACK standard. It is suggested that you specify  $lwork=0$ .
- If  $lwork = -1$ , PDPBTRF performs a work area query and returns the optimum required size of  $work$  in  $work_1$ . No computation is performed and the subroutine returns after error checking is complete.
- Otherwise, it must have the following value:  

$$lwork \geq k^2$$

*info* See On Return.

## On Return

*a*  $a$  is the updated local part of the global matrix  $A$ , containing the results of the factorization, where:

- If  $uplo = 'U'$ , the leading  $n \times n$  upper triangular part of the global submatrix  $A_{ja:ja+n-1, ja:ja+n-1}$  contains the results of the factorization. The remaining elements stored in submatrix  $A$  were overwritten by this subroutine.
- If  $uplo = 'L'$ , the leading  $n \times n$  lower triangular part of the global submatrix  $A_{ja:ja+n-1, ja:ja+n-1}$  contains the results of the factorization. The remaining elements stored in submatrix  $A$  were overwritten by this subroutine.

Scope: **local**

Returned as: an LLD\_A by (at least) LOCp( $ja+n-1$ ) array, containing numbers of the data type indicated in Table 83 on page 498.

On output, array  $A$  is overwritten; that is, original input is not preserved.

*af* is a reserved area.

*work* is the work area used by this subroutine if  $lwork \neq 0$ , where:

If  $lwork \neq 0$  or  $lwork \neq -1$ , the size of  $work$  is (at least) of length  $lwork$ .

If  $lwork = -1$ , the size of  $work$  is (at least) of length 1.

Scope: **local**

Returned as: an area of storage, containing numbers of the data type indicated in Table 83 on page 498, where:

- If  $lwork \geq 1$ ,  $work_1$  is set to the minimum  $lwork$  value needed.
- If  $lwork = -1$ ,  $work_1$  is set to the optimum  $lwork$  value needed.

Except for  $work_1$ , the contents of  $work$  are overwritten on return.

*info* has the following meaning:

If  $info = 0$ , global submatrix  $A$  is positive definite and the factorization completed normally, or the work area query completed successfully.

If  $info > 0$ , the leading minor of order  $i$  of the global submatrix  $A$  is not positive definite.  $info$  is set equal to  $i$ , where the first leading minor was encountered at  $A_{ja+i-1, ja+i-1}$ . The results contained in matrix  $A$  are not defined.

Scope: **global**

Returned as: a fullword integer;  $info \geq 0$ .

## Notes and Coding Rules

1. In your C program, argument *info* must be passed by reference.
2. This subroutine accepts lowercase letters for the *uplo* argument.
3. This subroutine gives the best performance for wide band widths, for example:

$$k > 100\sqrt{p}$$

where  $p$  is the number of processes. For details, see references [2], [41], and [42]. Also, it is suggested that you specify *uplo* = 'L'.

4. The  $k+1$  by  $n$  array specified for submatrix  $A$  must remain unchanged between calls to PDPBTRF and PDPBTRS. This subroutine overwrites data in positions that do not contain the positive definite symmetric band matrix  $A$  stored in upper- or lower-band-packed storage mode.
5. The output from this factorization subroutine should be used only as input to the solve subroutine PDPBTRS.

The data specified for input arguments *uplo*,  $n$ , and  $k$  must be the same for both PDPBTRF and PDPBTRS.

The matrix  $A$  and *af* input to PDPBTRS must be the same as the corresponding output arguments for PDPBTRF; and thus, the scalar data specified for *ja*, *desc\_a*, and *laf* must also be the same.

6. In all cases, follow these rules:
  - DTYPE\_A=501 or 1
  - If DTYPE\_A=1, RSRC\_A=0,  $M\_A \geq k+1$ , and  $MB\_A \geq 1$ .
  - Following are the allowable array descriptor types and process grids, where  $p$  is the number of processes in the process grid:

DTYPE_A	Process Grid
501	$p \times 1$ or $1 \times p$
1	$1 \times p$

7. To determine the values of  $LOCp(n)$  used in the argument descriptions, see "Determining the Number of Rows and Columns in Your Local Arrays" on page 28 for descriptor type-1 or "Determining the Number of Rows or Columns in Your Local Arrays" on page 32 for descriptor type-501 and type-502.
8. Matrix  $A$ , *af*, and *work* must have no common elements; otherwise, results are unpredictable.
9. The global symmetric band matrix  $A$  must be positive definite. If  $A$  is not positive definite, this subroutine uses the *info* argument to provide information about  $A$  and issues an error message. This differs from ScaLAPACK, which only uses the *info* argument to provide information about  $A$ .



10. The global positive definite symmetric band matrix  $A$  must be stored in upper- or lower-band-packed storage mode. See the section on block-cyclically distributing a symmetric matrix in “Matrices” on page 40.  
Matrix  $A$  must be distributed over a one-dimensional process grid, using block-cyclic data distribution. For more information on using block-cyclic data distribution, see “Specifying Block-Cyclically-Distributed Matrices for the Banded Linear Algebraic Equations” on page 29.
11. If  $lwork = -1$  on any process, it must equal  $-1$  on all processes. That is, if a subset of the processes specifies  $-1$  for the work area size, they must all specify  $-1$ .
12. Although global matrix  $A$  may be block-cyclically distributed on a  $1 \times p$  or  $p \times 1$  process grid, the values of  $n$ ,  $ja$ , and  $NB\_A$  must be chosen so that each process has at most one full or partial block of the global submatrix  $A$ .

## Error Conditions

### Computational Errors

Matrix  $A$  is not positive definite. For details, see the description of the *info* argument.

### Resource Errors

$lwork = 0$  and unable to allocate workspace

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1.  $DTYPE\_A$  is invalid.

#### Stage 2:

1.  $CTXT\_A$  is invalid.

#### Stage 3:

1. PDPBTRF was called from outside the process grid.

#### Stage 4:

1. The process grid is not  $1 \times p$  or  $p \times 1$ .
2.  $uplo \neq 'U'$  or  $'L'$
3.  $n < 0$
4.  $ja < 1$
5.  $k < 0$
6.  $k+1 > n$
7.  $DTYPE\_A = 1$  and:
  - a.  $M\_A < k+1$
  - b.  $MB\_A < 1$
  - c.  $RSRC\_A \neq 0$
  - d. The process grid is not  $1 \times p$ .
8.  $N\_A < 0$  and  $(n = 0)$ ;  $N\_A < 1$  otherwise
9.  $NB\_A < 1$
10.  $n > (NB\_A)p - \text{mod}(ja-1, NB\_A)$
11.  $CSRC\_A < 0$  or  $CSRC\_A \geq p$
12.  $uplo = 'U'$  and  $k > NB\_A$ .

#### Stage 5:

1.  $ja > N\_A$  and  $(n > 0)$
2.  $ja+n-1 > N\_A$  and  $(n > 0)$
3.  $LLD\_A < k+1$

**Stage 6:**

1.  $lwork \neq 0$ ,  $lwork \neq -1$ , and  $lwork < k^2$

**Stage 7:**

Each of the following global input arguments are checked to determine whether its value differs from the value specified on process  $P_{00}$ :

1.  $uplo$  differs.
2.  $n$  differs.
3.  $k$  differs.
4.  $ja$  differs.
5.  $DTYPE\_A$  differs.
6.  $DTYPE\_A$  does not differ and:
  - a.  $N\_A$  differs.
  - b.  $NB\_A$  differs.
  - c.  $CSRC\_A$  differs.
  - d.  $DTYPE\_A = 1$  and:
    - 1)  $M\_A$  differs.
    - 2)  $MB\_A$  differs.
    - 3)  $RSRC\_A$  differs.

Also:

7.  $lwork = -1$  on a subset of processes.

## Examples

### Example

This example shows a factorization of the positive definite symmetric band matrix  $A$  of order 9 with a half bandwidth of 7:

$$\begin{bmatrix} 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 0.0 \\ 1.0 & 2.0 & 2.0 & 2.0 & 2.0 & 2.0 & 2.0 & 2.0 & 1.0 \\ 1.0 & 2.0 & 3.0 & 3.0 & 3.0 & 3.0 & 3.0 & 3.0 & 2.0 \\ 1.0 & 2.0 & 3.0 & 4.0 & 4.0 & 4.0 & 4.0 & 4.0 & 3.0 \\ 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 5.0 & 5.0 & 5.0 & 4.0 \\ 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 6.0 & 6.0 & 6.0 & 5.0 \\ 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 6.0 & 7.0 & 7.0 & 6.0 \\ 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 6.0 & 7.0 & 8.0 & 7.0 \\ 0.0 & 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 6.0 & 7.0 & 8.0 \end{bmatrix}$$

Matrix  $A$  is stored in lower-band-packed storage mode:

$$\begin{bmatrix} 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 6.0 & 7.0 & 8.0 & 8.0 \\ 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 6.0 & 7.0 & 7.0 & . \\ 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 6.0 & 6.0 & . & . \\ 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 5.0 & . & . & . \\ 1.0 & 2.0 & 3.0 & 4.0 & 4.0 & . & . & . & . \\ 1.0 & 2.0 & 3.0 & 3.0 & . & . & . & . & . \\ 1.0 & 2.0 & 2.0 & . & . & . & . & . & . \\ 1.0 & 1.0 & . & . & . & . & . & . & . \end{bmatrix}$$

where “.” means you do not have to store a value in that position in the local array. However, these storage positions are required and are overwritten during the computation.

Matrix  $A$  is distributed over a  $1 \times 3$  process grid using block-cyclic distribution.

**Notes:**

1. Matrix  $A$ , output from PDPBTRF, must be passed, unchanged, to the solve subroutine PDPBTRS.
2. The  $laf$  argument must be specified; however, this subroutine currently ignores its value. For migration purposes, in this example,  $laf$  is specified as 119.
3. The  $af$  argument is reserved and not shown in this example.
4. Because  $lwork = 0$ , PDPBTRF dynamically allocates the work area used by this subroutine.

**Call Statements and Input:**

```

ORDER = 'R'
NPROW = 1
NPCOL = 3
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      UPLO  N   K   A   JA  DESC_A  AF   LAF  WORK LWORK INFO
      |    |   |   |   |   |      |   |   |   |   |
CALL PDPBTRF( 'L' , 9 , 7 , A , 1 , DESC_A , AF , 119 , WORK , 0 , INFO )

```

	Desc_A
DTYPE_	501
CTXT_	<i>icontxt</i> <sup>1</sup>
N_	9
NB_	3
CSRC_	0
LLD_A	8
Reserved	—

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.

Global matrix  $A$  stored in lower-band-packed storage mode with block size of 3:

$$\begin{array}{c}
 \text{B,D} \quad \quad \quad 0 \quad \quad \quad 1 \quad \quad \quad 2 \\
 \\
 \begin{array}{c} 0 \end{array} \left[ \begin{array}{ccc|ccc|ccc}
 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 6.0 & 7.0 & 8.0 & 8.0 \\
 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 6.0 & 7.0 & 7.0 & . \\
 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 6.0 & 6.0 & . & . \\
 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 5.0 & . & . & . \\
 1.0 & 2.0 & 3.0 & 4.0 & 4.0 & . & . & . & . \\
 1.0 & 2.0 & 3.0 & 3.0 & . & . & . & . & . \\
 1.0 & 2.0 & 2.0 & . & . & . & . & . & . \\
 1.0 & 1.0 & . & . & . & . & . & . & .
 \end{array} \right]
 \end{array}$$

The following is the  $1 \times 3$  process grid:

B,D	0	1	2
0	P <sub>00</sub>	P <sub>01</sub>	P <sub>02</sub>

Local array  $A$  with block size of 3:

p,q	0	1	2
	1.0 2.0 3.0	4.0 5.0 6.0	7.0 8.0 8.0
	1.0 2.0 3.0	4.0 5.0 6.0	7.0 7.0 .

## PDPBTRF

0	1.0	2.0	3.0	4.0	5.0	6.0	6.0	.	.
	1.0	2.0	3.0	4.0	5.0	5.0	.	.	.
	1.0	2.0	3.0	4.0	4.0	.	.	.	.
	1.0	2.0	3.0	3.0	.	.	.	.	.
	1.0	2.0	2.0	.	.	.	.	.	.
	1.0	1.0	.	.	.	.	.	.	.

### Output:

Global matrix  $A$  is returned in lower-band-packed storage mode with block size of 3:

B,D	0			1			2		
0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	.
	1.0	1.0	1.0	1.0	1.0	1.0	1.0	.	.
	1.0	1.0	1.0	1.0	1.0	1.0	.	.	.
	1.0	1.0	1.0	1.0	1.0	.	.	.	.
	1.0	1.0	1.0	1.0	.	.	.	.	.
	1.0	1.0	1.0	.	.	.	.	.	.
	1.0	1.0	.	.	.	.	.	.	.

The following is the  $1 \times 3$  process grid:

B,D	0	1	2
0	P <sub>00</sub>	P <sub>01</sub>	P <sub>02</sub>

Local array  $A$  with block size of 3:

p,q	0			1			2		
0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	.
	1.0	1.0	1.0	1.0	1.0	1.0	1.0	.	.
	1.0	1.0	1.0	1.0	1.0	1.0	.	.	.
	1.0	1.0	1.0	1.0	1.0	.	.	.	.
	1.0	1.0	1.0	1.0	.	.	.	.	.
	1.0	1.0	1.0	.	.	.	.	.	.
	1.0	1.0	.	.	.	.	.	.	.

The value of *info* is 0 on all processes.

## PDPBTRS — Positive Definite Symmetric Band Matrix Solve

### Purpose

This subroutine solves the following system of equations for multiple right-hand sides:

- $AX = B$

where, in the formula above:

- $A$  represents the global positive definite symmetric band submatrix  $A_{ja:ja+n-1, ja:ja+n-1}$  factored by Cholesky factorization.
- $B$  represents the global general submatrix  $B_{ib:ib+n-1, 1:nrhs}$  containing the right-hand sides in its columns.
- $X$  represents the global general submatrix  $B_{ib:ib+n-1, 1:nrhs}$  containing the output solution vectors in its columns.

This subroutine uses the results of the factorization of matrix  $A$ , produced by a preceding call to PDPBTRF. The output from PDPBTRF should be used only as input to this solve subroutine.

If  $n = 0$  or  $nrhs = 0$ , no computation is performed and the subroutine returns after doing some parameter checking. See references [2], [24], [41], and [42].

Table 84. Data Types

$A, B, af, work$	Subroutine
Long-precision real	PDPBTRS

### Syntax

<b>Fortran</b>	CALL PDPBTRS ( <i>uplo, n, k, nrhs, a, ja, desc_a, b, ib, desc_b, af, laf, work, lwork, info</i> )
<b>C and C++</b>	pdpbtrs ( <i>uplo, n, k, nrhs, a, ja, desc_a, b, ib, desc_b, af, laf, work, lwork, info</i> );

### On Entry

*uplo* indicates whether the upper or lower triangular part of the global submatrix  $A$  is referenced, where:

If *uplo* = 'U', the upper triangular part is referenced.

If *uplo* = 'L', the lower triangular part is referenced.

Scope: **global**

Specified as: a single character; *uplo* = 'U' or 'L'.

*n* is the number of columns in the submatrix  $A$ , stored in the upper- or lower-band-packed storage mode. It is also the number of rows in the general submatrix  $B$  containing the multiple right-hand sides.

Scope: **global**

Specified as: a fullword integer;  $0 \leq n \leq (\text{NB\_A})p - \text{mod}(ja-1, \text{NB\_A})$ .

*k* is the half bandwidth of the factored submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer, where:

- If *uplo* = 'U',  $0 \leq k \leq \text{NB\_A}$ .

- If  $uplo = 'L', 0 \leq k < n$ .

These limits for  $k$  are extensions of the ScaLAPACK standard.

*nrhs* is the number of columns in submatrix  $B$  used in the computation.

Scope: **global**

Specified as: a fullword integer;  $nrhs \geq 0$ .

*a* is the local part of the global positive definite symmetric band matrix  $A$ , stored in upper- or lower-band-packed storage mode, containing the factorization of matrix  $A$  produced from a preceding call to PDPBTRF. This identifies the **first element** of the local array  $A$ . This subroutine computes the location of the first element of the local subarray used, based on  $k$ ,  $ja$ ,  $desc\_a$ , and  $p$ ; therefore, the leading  $k+1$  by  $LOCp(ja+n-1)$  part of the local array  $A$  must contain the local pieces of the leading  $k+1$  by  $ja+n-1$  part of the global matrix, and:

- If  $uplo = 'U'$ , the leading  $n \times n$  upper triangular part of the global submatrix  $A_{ja:ja+n-1, ja:ja+n-1}$  contains the factorization.
- If  $uplo = 'L'$ , the leading  $n \times n$  lower triangular part of the global submatrix  $A_{ja:ja+n-1, ja:ja+n-1}$  contains the factorization.

Scope: **local**

Specified as: an LLD\_A by (at least)  $LOCp(ja+n-1)$  array, containing numbers of the data type indicated in Table 84 on page 507. Details about the block-cyclic data distribution of global matrix  $A$  are stored in  $desc\_a$ .

On output, array  $A$  is overwritten; that is, original input is not preserved.

*ja* is the column index of the global matrix  $A$ , identifying the first column of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$  and  $ja+n-1 \leq N\_A$ .

*desc\_a* is the array descriptor for global matrix  $A$ , which may be type 501 or type 1, as described in the following tables. For rules on using array descriptors, see "Notes and Coding Rules" on page 512.

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor Type	DTYPE_A = 501 for $1 \times p$ or $p \times 1$  where $p$ is the number of processes in a process grid.	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	N_A	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
4	NB_A	Column block size	$NB\_A \geq 1$ and $0 \leq n \leq (NB\_A)p - \text{mod}(ja-1, NB\_A)$	Global
5	CSRC_A	The process column over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < p$	Global
6	LLD_A	Leading dimension	$LLD\_A \geq k+1$	<b>Local</b>

<i>desc_a</i>	Name	Description	Limits	Scope
7	—	Reserved	—	—

Specified as: an array of (at least) length 7, containing fullword integers.

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A = 1 for $1 \times p$ where $p$ is the number of processes in a process grid.	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	$M\_A > k$	Global
4	N_A	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$ and $0 \leq n \leq (NB\_A)p - \text{mod}(ja-1, NB\_A)$	Global
7	RSRC_A	The process row over which the first row of the global matrix is distributed	$RSRC\_A = 0$	Global
8	CSRC_A	The process column over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < p$	Global
9	LLD_A	The leading dimension of the local array	$LLD\_A \geq k+1$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

*b* is the local part of the global general matrix **B**, containing the multiple right-hand sides of the system. This identifies the **first element** of the local array **B**. This subroutine computes the location of the first element of the local subarray used, based on *ib*, *desc\_b*, and *p*; therefore, the leading  $\text{LOCp}(ib+n-1)$  by *nrhs* part of the local array **B** must contain the local pieces of the leading  $ib+n-1$  by *nrhs* part of the global matrix.

Scope: **local**

Specified as: an LLD\_B by (at least) *nrhs* array, containing numbers of the data type indicated in Table 84 on page 507. Details about the block-cyclic data distribution of global matrix **B** are stored in *desc\_b*.

*ib* is the row index of the global matrix **B**, identifying the first row of the submatrix **B**.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ib \leq M\_B$ .

*desc\_b* is the array descriptor for global matrix **B**, which may be type 502 or type 1, as described in the following tables. For rules on using array descriptors, see “Notes and Coding Rules” on page 512.

## PDPBTRS

<i>desc_b</i>	Name	Description	Limits	Scope
1	DTYPE_B	Descriptor type	DTYPE_B = 502 for $p \times 1$ or $1 \times p$ where $p$ is the number of processes in a process grid.	Global
2	CTXT_B	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_B	Number of rows in the global matrix	If $n = 0$ : M_B $\geq 0$ Otherwise: M_B $\geq 1$	Global
4	MB_B	Row block size	MB_B $\geq 1$ and $0 \leq n \leq (MB\_B)p - \text{mod}(ib-1, MB\_B)$	Global
5	RSRC_B	The process row over which the first row of the global matrix is distributed	$0 \leq \text{RSRC\_B} < p$	Global
6	LLD_B	Leading dimension	LLD_B $\geq \max(1, \text{LOCp}(M\_B))$	Local
7	—	Reserved	—	—

Specified as: an array of (at least) length 7, containing fullword integers.

<i>desc_b</i>	Name	Description	Limits	Scope
1	DTYPE_B	Descriptor type	DTYPE_B = 1 for $p \times 1$ where $p$ is the number of processes in a process grid.	Global
2	CTXT_B	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_B	Number of rows in the global matrix	If $n = 0$ : M_B $\geq 0$ Otherwise: M_B $\geq 1$	Global
4	N_B	Number of columns in the global matrix	N_B $\geq nrhs$	Global
5	MB_B	Row block size	MB_B $\geq 1$ and $0 \leq n \leq (MB\_B)p - \text{mod}(ib-1, MB\_B)$	Global
6	NB_B	Column block size	NB_B $\geq 1$	Global
7	RSRC_B	The process row over which the first row of the global matrix is distributed	$0 \leq \text{RSRC\_B} < p$	Global
8	CSRC_B	The process column over which the first column of the global matrix is distributed	CSRC_B=0	Global
9	LLD_B	Leading dimension	LLD_B $\geq \max(1, \text{LOCp}(M\_B))$	Local

Specified as: an array of (at least) length 9, containing fullword integers.

*af* is a reserved area. Its size is specified by LAF.

Scope: **local**



Specified as: for migration purposes, you should specify a one-dimensional, long-precision array of (at least) length  $laf$ .

$laf$  is the number of elements in array AF.

The  $laf$  argument must be specified; however, this subroutine currently ignores its value. For migration purposes, you should specify  $laf$  using the formula below.

Scope: **local**

Specified as: a fullword integer,  $laf \geq (NB\_A+2k)(k)$ .

$work$  has the following meaning:

If  $lwork = 0$ ,  $work$  is ignored.

If  $lwork \neq 0$ ,  $work$  is the work area used by this subroutine, where:

- If  $lwork \neq -1$ , the size of  $work$  is (at least) of length  $lwork$ .
- If  $lwork = -1$ , the size of  $work$  is (at least) of length 1.

Scope: **local**

Specified as: an area of storage containing numbers of data type indicated in Table 84 on page 507.

$lwork$  is the number of elements in array WORK.

Scope:

- If  $lwork \geq 0$ ,  $lwork$  is **local**
- If  $lwork = -1$ ,  $lwork$  is **global**

Specified as: a fullword integer; where:

- If  $lwork = 0$ , PDPBTRS dynamically allocates the work area used by the subroutine. The work area is deallocated before control is returned to the calling program. This option is an extension to the ScaLAPACK standard. It is suggested that you specify  $lwork=0$ .
- If  $lwork = -1$ , PDPBTRS performs a work area query and returns the optimum required size of  $work$  in  $work_1$ . No computation is performed and the subroutine returns after error checking is complete.
- Otherwise,  

$$lwork \geq (nrhs)(k)$$

$info$  See On Return.

## On Return

$b$   $b$  is the updated local part of the global matrix  $B$ , containing the solution vectors.

Scope: **local**

Returned as: an LLD\_B by (at least)  $nrhs$  array, containing numbers of the data type indicated in Table 84 on page 507.

$work$  is the work area used by this subroutine if  $lwork \neq 0$ , where:

If  $lwork \neq 0$  or  $lwork \neq -1$ , the size of  $work$  is (at least) of length  $lwork$ .

If  $lwork = -1$ , the size of  $work$  is (at least) of length 1.

Scope: **local**

Returned as: an area of storage, containing numbers of the data type indicated in Table 84 on page 507, where:

- If  $lwork = -1$ ,  $work_1$  is set to the optimum  $lwork$  value needed.
- If  $lwork \geq 1$ ,  $work_1$  is set to the minimum  $lwork$  value needed.

Except for  $work_1$ , the contents of  $work$  are overwritten on return.

*info* indicates a successful computation or work area query occurred.

Scope: **global**

Returned as: a fullword integer; *info* = 0.

## Notes and Coding Rules

1. In your C program, argument *info* must be passed by reference.
2. The subroutine accepts lowercase letters for the *uplo* argument.
3. This subroutine gives the best performance for wide band widths, for example:

$$k > 100\sqrt{p}$$

where  $p$  is the number of processes). For details, see references [2], [41], and [42]. Also, it is suggested that you specify *uplo* = 'L'.

4. The  $k+1$  by  $n$  array specified for submatrix  $A$  must remain unchanged between calls to PDPBTRF and PDPBTRS. This subroutine overwrites data in positions that do not contain the positive definite symmetric band matrix  $A$  stored in upper- or lower-band-packed storage mode.
5. The output from the PDPBTRF subroutine should be used only as input to the solve subroutine PDPBTRS.

The input arguments *uplo*,  $n$ , and  $k$  must be the same for both PDPBTRF and PDPBTRS.

The global matrix  $A$  and *af* input to PDPBTRS must be the same as the corresponding output arguments for PDPBTRF; and thus, the scalar data specified for *ja*, *desc\_a*, and *laf* must also be the same.

6. In all cases, follow these rules:
  - $ib = ja$
  - DTYPE\_A=501 or 1
  - DTYPE\_B=502 or 1
  - NB\_A = MB\_B
  - If DTYPE\_A=1, RSRC\_A=0,  $M_A \geq k+1$ , and  $MB_A \geq 1$ .
  - If DTYPE\_B=1, CSRC\_B=0,  $N_B \geq nrhs$ , and  $NB_B \geq 1$ .
  - CTXT\_A = CTXT\_B
  - Following are the consistent combinations of array descriptor types and process grids, where  $p$  is the number of processes in the process grid:

DTYPE_A	DTYPE_B	Process Grid
501	502	$p \times 1$ or $1 \times p$
501	1	$1 \times p$
1	502	$p \times 1$
1	1	$1 \times 1$

7. To determine the values of LOCp( $n$ ) used in the argument descriptions, see "Determining the Number of Rows and Columns in Your Local Arrays" on page 28

page 28 for descriptor type-1 or “Determining the Number of Rows or Columns in Your Local Arrays” on page 32 for descriptor type-501 and type-502.

8.  $A$ ,  $B$ ,  $af$  and  $work$  must have no common elements; otherwise, results are unpredictable.
9. The global positive definite symmetric band matrix  $A$  must be stored in upper- or lower-band-packed storage mode. See the section on block distributing a symmetric matrix in “Matrices” on page 40.  
Matrix  $A$  must be distributed over a one-dimensional process grid, using block-cyclic data distribution. For more information on using block-cyclic data distribution, see “Specifying Block-Cyclically-Distributed Matrices for the Banded Linear Algebraic Equations” on page 29.
10. Matrix  $B$  must be distributed over a one-dimensional process grid, using block-cyclic data distribution. For more information on using block-cyclic data distribution, see “Specifying Block-Cyclically-Distributed Matrices for the Banded Linear Algebraic Equations” on page 29. Also, see the section on distributing the right-hand side matrix in “Matrices” on page 40.
11. If  $lwork = -1$  on any process, it must equal  $-1$  on all processes. That is, if a subset of the processes specifies  $-1$ , they must all specify  $-1$ .
12. Although global submatrices  $A$  and  $B$  may be block-cyclically distributed on a  $1 \times p$  or  $p \times 1$  process grid, the values of  $n$ ,  $ja$ ,  $ib$ ,  $NB\_A$ , and  $MB\_B$  must be chosen so that each process has at most one full or partial block of each of the global submatrices  $A$  and  $B$ .

## Error Conditions

### Computational Errors

None

**Note:** If the factorization performed by PDPBTRF failed because of a nonpositive definite matrix  $A$ , the results returned by this subroutine are unpredictable. For details, see the *info* output argument for PDPBTRF.

### Resource Errors

$lwork = 0$  and unable to allocate workspace

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1.  $DTYPE\_A$  is invalid.
2.  $DTYPE\_B$  is invalid.

#### Stage 2:

1.  $CTXT\_A$  is invalid.

#### Stage 3:

1. PDPBTRS was called from outside the process grid.

#### Stage 4:

1. The process grid is not  $1 \times p$  or  $p \times 1$ .
2.  $uplo \neq 'U'$  or  $'L'$
3.  $n < 0$
4.  $k < 0$
5.  $k+1 > n$
6.  $ja < 1$

7. DTYPE\_A = 1 and:
  - a. M\_A < k+1
  - b. MB\_A < 1
  - c. RSRC\_A ≠ 0
  - d. The process grid is not 1 × p.
8. N\_A < 0 and (n = 0); N\_A < 1 otherwise
9. NB\_A < 1
10. n > (NB\_A)p-mod(ja-1,NB\_A)
11. uplo = 'U' and k > NB\_A
12. CSRC\_A < 0 or CSRC\_A ≥ p
13. nrhs < 0
14. ib ≠ ja
15. ib < 1
16. DTYPE\_B = 1 and:
  - a. N\_B < nrhs
  - b. NB\_B < 1
  - c. CSRC\_B ≠ 0
  - d. The process grid is not p × 1.
17. M\_B < 0 and (n = 0); M\_B < 1 otherwise
18. MB\_B < 1
19. n > (MB\_B)p-mod(ib-1,MB\_B)
20. MB\_B ≠ NB\_A
21. RSRC\_B < 0 or RSRC\_B ≥ p
22. CTXT\_A ≠ CTXT\_B

**Stage 5:** If n > 0:

1. ja+n-1 > N\_A
2. ja > N\_A
3. ib > M\_B
4. ib+n-1 > M\_B
5. LLD\_A < k+1

**Stage 6:**

1. LLD\_B < max(1, LOCp(M\_B))
2. lwork ≠ 0,
3. lwork ≠ -1, and lwork < (nrhs)(k)

**Stage 7:**

Each of the following global input arguments are checked to determine whether its value differs from the value specified on process P<sub>00</sub>:

1. uplo differs.
2. n differs.
3. k differs.
4. nrhs differs.
5. ja differs.
6. DTYPE\_A differs.
7. DTYPE\_A does not differ and:
  - a. N\_A differs.
  - b. NB\_A differs.
  - c. CSRC\_A differs.
  - d. DTYPE\_A = 1 and:
    - 1) M\_A differs.
    - 2) MB\_A differs.
    - 3) RSRC\_A differs.
8. ib differs.

9. DTYPE\_B differs.
  10. DTYPE\_B does not differ and:
    - a. M\_B differs.
    - b. MB\_B differs.
    - c. RSRC\_B differs.
    - d. DTYPE\_A = 1 and:
      - 1) N\_B differs.
      - 2) NB\_B differs.
      - 3) CSRC\_B differs.
- Also:
11. *lwork* = -1 on a subset of processes.

## Examples

### Example

This example solves the  $AX=B$  system, where matrix  $A$  is the same positive definite symmetric band matrix factored in “Example” on page 504 for PDPBTRE.

#### Notes:

1. Matrix  $A$ , output from PDPBTRE, must be passed, unchanged, to the solve subroutine PDPBTRS.  
The input values for *desc\_a* are the same values shown in “Example” on page 504.
2. Notice **only one process grid was created**, even though, DTYPE\_A = 501 and DTYPE\_B = 502.
3. The *laf* argument must be specified; however, this subroutine currently ignores its value. For migration purposes, in this example, *laf* is specified as 119.
4. The *af* argument, output from PDPBTRE, must be passed, unchanged, to the solve subroutine PDPBTRS.
5. Because *lwork* = 0, PDPBTRS dynamically allocates the work area used by this subroutine.

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 1
NPCOL = 3
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      UPLO  N   K  NRHS A   JA  DESC_A  B   IB  DESC_B  AF  LAF
      |    |   |   |   |   |         |   |   |         |   |
CALL PDPBTRS( 'L' , 9 , 7 , 3 , A , 1 , DESC_A , B , 1 , DESC_B , AF , 119 ,

      WORK LWORK INFO
      |    |    |
      WORK , 0 , INFO )
```

	Desc_B
DTYPE_	502
CTXT_	<i>icontxt</i> <sup>1</sup>
M_	9
MB_	3
RSRC_	0

	Desc_B
LLD_B	3
Reserved	—

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.

Global matrix *A* stored in lower-band-packed storage mode with block size of 3:

B,D	0	1	2
0	$\begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & . \end{bmatrix}$	$\begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & . \\ 1.0 & . & . \\ . & . & . \\ . & . & . \end{bmatrix}$	$\begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & . \\ 1.0 & . & . \\ . & . & . \\ . & . & . \\ . & . & . \\ . & . & . \\ . & . & . \end{bmatrix}$

The following is the  $1 \times 3$  process grid:

B,D	0	1	2
0	P <sub>00</sub>	P <sub>01</sub>	P <sub>02</sub>

Local array *A* with block size of 3:

p,q	0	1	2
0	$\begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & . \end{bmatrix}$	$\begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & . \\ 1.0 & . & . \\ . & . & . \\ . & . & . \end{bmatrix}$	$\begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & . \\ 1.0 & . & . \\ . & . & . \\ . & . & . \\ . & . & . \\ . & . & . \\ . & . & . \end{bmatrix}$

Global matrix *B* with block size of 3:

B,D	0
0	$\begin{bmatrix} 8.0 & 36.0 & 44.0 \\ 16.0 & 80.0 & 80.0 \\ 23.0 & 122.0 & 108.0 \end{bmatrix}$
1	$\begin{bmatrix} 29.0 & 161.0 & 129.0 \\ 34.0 & 196.0 & 144.0 \\ 38.0 & 226.0 & 154.0 \end{bmatrix}$
2	$\begin{bmatrix} 41.0 & 250.0 & 160.0 \\ 43.0 & 267.0 & 163.0 \\ 36.0 & 240.0 & 120.0 \end{bmatrix}$

The following is the  $1 \times 3$  process grid:

B,D	0	1	2
0	P <sub>00</sub>	P <sub>01</sub>	P <sub>02</sub>

Local array *B* with block size of 3:

p,q	0	1	2
0	8.0 36.0 44.0 16.0 80.0 80.0 23.0 122.0 108.0	29.0 161.0 129.0 34.0 196.0 144.0 38.0 226.0 154.0	41.0 250.0 160.0 43.0 267.0 163.0 36.0 240.0 120.0

Output:

Global matrix *B* with block size of 3:

B,D	0
0	1.0 1.0 9.0 1.0 2.0 8.0 1.0 3.0 7.0
1	1.0 4.0 6.0 1.0 5.0 5.0 1.0 6.0 4.0
2	1.0 7.0 3.0 1.0 8.0 2.0 1.0 9.0 1.0

The following is the 1 × 3 process grid:

B,D	0	1	2
0	P <sub>00</sub>	P <sub>01</sub>	P <sub>02</sub>

Local array B with block size of 3:

p,q	0	1	2
0	1.0 1.0 9.0 1.0 2.0 8.0 1.0 3.0 7.0	1.0 4.0 6.0 1.0 5.0 5.0 1.0 6.0 4.0	1.0 7.0 3.0 1.0 8.0 2.0 1.0 9.0 1.0

The value of *info* is 0 on all processes.

## PDGTSV and PDDTSV — General Tridiagonal Matrix Factorization and Solve

### Purpose

PDGTSV solves the tridiagonal systems of linear equations,  $AX = B$ , using Gaussian elimination with partial pivoting for the general tridiagonal matrix  $A$  stored in tridiagonal storage mode.

PDDTSV solves the tridiagonal systems of linear equations,  $AX = B$ , using Gaussian elimination for the diagonally dominant general tridiagonal matrix  $A$  stored in tridiagonal storage mode.

- $A$  represents the global square general tridiagonal submatrix  $A_{ia:ia+n-1, ia:ia+n-1}$ .
- $B$  represents the global general submatrix  $B_{ib:ib+n-1, 1:nrhs}$  containing the right-hand sides in its columns.
- $X$  represents the global general submatrix  $B_{ib:ib+n-1, 1:nrhs}$  containing the output solution vectors in its columns.

If  $n$  is 0, no computation is performed and the subroutine returns after doing some parameter checking. If  $n > 0$  and  $nrhs$  is 0, no solutions are computed and the subroutine returns after factoring the matrix.

See reference [54].

Table 85. Data Types

<i>dl, d, du, B, work</i>	Subroutine
Long-precision real	PDGTSV and PDDTSV

### Syntax

Fortran	CALL PDGTSV   PDDTSV ( <i>n, nrhs, dl, d, du, ia, desc_a, b, ib, desc_b, work, lwork, info</i> )
C and C++	pdgtsv   pddtsv ( <i>n, nrhs, dl, d, du, ia, desc_a, b, ib, desc_b, work, lwork, info</i> );

### On Entry

$n$  is the order of the general tridiagonal matrix  $A$  and the number of rows in the general submatrix  $B$ , which contains the multiple right-hand sides.

Scope: **global**

Specified as: a fullword integer, where:

- If (the process grid is  $p \times 1$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 502$ ,  $0 \leq n \leq (MB\_A)(p) - \text{mod}(ia-1, MB\_A)$ .
- If (the process grid is  $1 \times p$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 501$ ,  $0 \leq n \leq (NB\_A)(p) - \text{mod}(ia-1, NB\_A)$ .

where  $p$  is the number of processes in a process grid.

$nrhs$  is the number of right-hand sides; that is, the number of columns in submatrix  $B$  used in the computation.

Scope: **global**

Specified as: a fullword integer;  $nrhs \geq 0$ .

$dl$  is the local part of the global vector  $dl$ . This identifies the **first element** of



the local array *DL*. These subroutines compute the location of the first element of the local subarray used, based on *ia*, *desc\_a*, and *p*; therefore, the leading  $\text{LOCp}(ia+n-1)$  part of the local array *DL* contains the local pieces of the leading  $ia+n-1$  part of the global vector.

The global vector *dl* contains the subdiagonal of the global general tridiagonal submatrix *A* in elements  $ia+1$  through  $ia+n-1$ .

Scope: **local**

Specified as: a one-dimensional array of (at least) length  $\text{LOCp}(ia+n-1)$ , containing numbers of the data type indicated in Table 85 on page 518. Details about block-cyclic data distribution of global matrix *A* are stored in *desc\_a*.

On output, *DL* is overwritten; that is, the original input is not preserved.

*d* is the local part of the global vector *d*. This identifies the **first element** of the local array *D*. These subroutines compute the location of the first element of the local subarray used, based on *ia*, *desc\_a*, and *p*; therefore, the leading  $\text{LOCp}(ia+n-1)$  part of the local array *D* contains the local pieces of the leading  $ia+n-1$  part of the global vector.

The global vector *d* contains the main diagonal of the global general tridiagonal submatrix *A* in elements  $ia$  through  $ia+n-1$ .

Scope: **local**

Specified as: a one-dimensional array of (at least) length  $\text{LOCp}(ia+n-1)$  containing numbers of the data type indicated in Table 85 on page 518. Details about block-cyclic data distribution of global matrix *A* are stored in *desc\_a*.

On output, *D* is overwritten; that is, the original input is not preserved.

*du* is the local part of the global vector *du*. This identifies the **first element** of the local array *DU*. These subroutines compute the location of the first element of the local subarray used, based on *ia*, *desc\_a*, and *p*; therefore, the leading  $\text{LOCp}(ia+n-1)$  part of the local array *DU* contains the local pieces of the leading  $ia+n-1$  part of the global vector.

The global vector *du* contains the superdiagonal of the global general tridiagonal submatrix *A* in elements  $ia$  through  $ia+n-2$ .

Scope: **local**

Specified as: a one-dimensional array of (at least) length  $\text{LOCp}(ia+n-1)$ , containing numbers of the data type indicated in Table 85 on page 518. Details about block-cyclic data distribution of global matrix *A* are stored in *desc\_a*.

On output, *DU* is overwritten; that is, the original input is not preserved.

*ia* is the row or column index of the global matrix *A*, identifying the first row or column of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer, where:

- If (the process grid is  $p \times 1$  and  $\text{DTYPE\_A} = 1$ ) or  $\text{DTYPE\_A} = 502$ ,  $1 \leq ia \leq M\_A$  and  $ia+n-1 \leq M\_A$ .
- If (the process grid is  $1 \times p$  and  $\text{DTYPE\_A} = 1$ ) or  $\text{DTYPE\_A} = 501$ ,  $1 \leq ia \leq N\_A$  and  $ia+n-1 \leq N\_A$ .

*desc\_a* is the array descriptor for global matrix *A*. Because vectors are one-dimensional data structures, you may use a type-502, type-501, or type-1 array descriptor regardless of whether the process grid is  $p \times 1$  or  $1 \times p$ . For a type-502 array descriptor, the process grid is used as if it is a  $p \times 1$  process grid. For a type-501 array descriptor, the process grid is used as if it is a  $1 \times p$  process grid. For a type-1 array descriptor, the process grid is used as if it is either a  $p \times 1$  process grid or a  $1 \times p$  process grid.

The following tables describe the three types of array descriptors. For rules on using array descriptors, see “Notes and Coding Rules” on page 525.

Table 86. Type-502 Array Descriptor

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor Type	DTYPE_A=502 for $p \times 1$ or $1 \times p$ where $p$ is the number of processes in a process grid.	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $n = 0$ : M_A $\geq 0$ Otherwise: M_A $\geq 1$	Global
4	MB_A	Row block size	MB_A $\geq 1$ and $0 \leq n \leq (\text{MB\_A})(p) - \text{mod}(ia-1, \text{MB\_A})$	Global
5	RSRC_A	The process row over which the first row of the global matrix is distributed	$0 \leq \text{RSRC\_A} < p$	Global
6	—	Not used by these subroutines.	—	—
7	—	Reserved	—	—

Specified as: an array of (at least) length 7, containing fullword integers.

Table 87. Type-1 Array Descriptor ( $p \times 1$  Process Grid)

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor Type	DTYPE_A = 1 for $p \times 1$ where $p$ is the number of processes in a process grid.	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $n = 0$ : M_A $\geq 0$ Otherwise: M_A $\geq 1$	Global
4	N_A	Number of columns in the global matrix	N_A = 1	
5	MB_A	Row block size	MB_A $\geq 1$ and $0 \leq n \leq (\text{MB\_A})(p) - \text{mod}(ia-1, \text{MB\_A})$	Global
6	NB_A	Column block size	NB_A $\geq 1$	Global

Table 87. Type-1 Array Descriptor ( $p \times 1$  Process Grid) (continued)

<i>desc_a</i>	Name	Description	Limits	Scope
7	RSRC_A	The process row over which the first row of the global matrix is distributed	$0 \leq \text{RSRC\_A} < p$	Global
8	CSRC_A	The process column over which the first column of the global matrix is distributed	$\text{CSRC\_A} = 0$	Global
9	—	Not used by these subroutines.	—	—

Specified as: an array of (at least) length 9, containing fullword integers.

Table 88. Type-501 Array Descriptor

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor Type	$\text{DTYPE\_A} = 501$ for $1 \times p$ or $p \times 1$ where $p$ is the number of processes in a process grid.	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	N_A	Number of columns in the global matrix	If $n = 0$ : $\text{N\_A} \geq 0$ Otherwise: $\text{N\_A} \geq 1$	Global
4	NB_A	Column block size	$\text{NB\_A} \geq 1$ and $0 \leq n \leq (\text{NB\_A})(p) - \text{mod}(ia-1, \text{NB\_A})$	Global
5	CSRC_A	The process column over which the first column of the global matrix is distributed	$0 \leq \text{CSRC\_A} < p$	Global
6	—	Not used by these subroutines.	—	—
7	—	Reserved	—	—

Specified as: an array of (at least) length 7, containing fullword integers.

Table 89. Type-1 Array Descriptor ( $1 \times p$  Process Grid)

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	$\text{DTYPE\_A} = 1$ for $1 \times p$ where $p$ is the number of processes in a process grid.	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	$\text{M\_A} = 1$	Global
4	N_A	Number of columns in the global matrix	If $n = 0$ : $\text{N\_A} \geq 0$ Otherwise: $\text{N\_A} \geq 1$	Global

Table 89. Type-1 Array Descriptor ( $1 \times p$  Process Grid) (continued)

<i>desc_a</i>	Name	Description	Limits	Scope
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$ and $0 \leq n \leq (NB\_A)(p) - \text{mod}(ia-1, NB\_A)$	Global
7	RSRC_A	The process row over which the first row of the global matrix is distributed	$RSRC\_A = 0$	Global
8	CSRC_A	The process column over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < p$	Global
9	—	Not used by these subroutines.	—	—

Specified as: an array of (at least) length 9, containing fullword integers.

- b* is the local part of the global general matrix **B**, containing the multiple right-hand sides of the system. This identifies the **first element** of the local array **B**. This subroutine computes the location of the first element of the local subarray used, based on *ib*, *desc\_b*, and *p*; therefore, the leading  $LOCp(ib+n-1)$  by *nrhs* part of the local array **B** must contain the local pieces of the leading *ib+n-1* by *nrhs* part of the global matrix.

Scope: **local**

Specified as: an LLD\_B by (at least) *nrhs* array, containing numbers of the data type indicated in Table 85 on page 518. Details about the block-cyclic data distribution of global matrix **B** are stored in *desc\_b*.

- ib* is the row index of the global matrix **B**, identifying the first row of the submatrix **B**.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ib \leq M\_B$  and  $ib+n-1 \leq M\_B$

- desc\_b* is the array descriptor for global matrix **B**, which may be type-502 or type-1, as described in the following tables. For type-502 array descriptor, the process grid is used as if it is a  $p \times 1$  process grid. For rules on using array descriptors, see “Notes and Coding Rules” on page 525.

<i>desc_b</i>	Name	Description	Limits	Scope
1	DTYPE_B	Descriptor type	$DTYPE\_B = 502$ for $p \times 1$ or $1 \times p$  where <i>p</i> is the number of processes in a process grid.	Global
2	CTXT_B	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_B	Number of rows in the global matrix	If $n = 0$ : $M\_B \geq 0$ Otherwise: $M\_B \geq 1$	Global
4	MB_B	Row block size	$MB\_B \geq 1$ and $0 \leq n \leq (MB\_B)p - \text{mod}(ib-1, MB\_B)$	Global

<i>desc_b</i>	Name	Description	Limits	Scope
5	RSRC_B	The process row over which the first row of the global matrix is distributed	$0 \leq \text{RSRC\_B} < p$	Global
6	LLD_B	Leading dimension	$\text{LLD\_B} \geq \max(1, \text{LOCp}(\text{M\_B}))$	Local
7	—	Reserved	—	—

Specified as: an array of (at least) length 7, containing fullword integers.

<i>desc_b</i>	Name	Description	Limits	Scope
1	DTYPE_B	Descriptor type	$\text{DTYPE\_B} = 1$ for $p \times 1$ where $p$ is the number of processes in a process grid.	Global
2	CTXT_B	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_B	Number of rows in the global matrix	If $n = 0$ : $\text{M\_B} \geq 0$ Otherwise: $\text{M\_B} \geq 1$	Global
4	N_B	Number of columns in the global matrix	$\text{N\_B} \geq \text{nrhs}$	Global
5	MB_B	Row block size	$\text{MB\_B} \geq 1$ and $0 \leq n \leq (\text{MB\_B})p - \text{mod}(ib-1, \text{MB\_B})$	Global
6	NB_B	Column block size	$\text{NB\_B} \geq 1$	Global
7	RSRC_B	The process row over which the first row of the global matrix is distributed	$0 \leq \text{RSRC\_B} < p$	Global
8	CSRC_B	The process column over which the first column of the global matrix is distributed	$\text{CSRC\_B} = 0$	Global
9	LLD_B	Leading dimension	$\text{LLD\_B} \geq \max(1, \text{LOCp}(\text{M\_B}))$	Local

Specified as: an array of (at least) length 9, containing fullword integers.

*work* has the following meaning:

If  $lwork = 0$ , *work* is ignored.

If  $lwork \neq 0$ , *work* is the work area used by this subroutine, where:

- If  $lwork \neq -1$ , the size of *work* is (at least) of length *lwork*.
- If  $lwork = -1$ , the size of *work* is (at least) of length 1.

Scope: **local**

Specified as: an area of storage containing numbers of data type indicated in Table 85 on page 518.

*lwork* is the number of elements in array WORK.

Scope:

- If  $lwork \geq 0$ , *lwork* is **local**
- If  $lwork = -1$ , *lwork* is **global**

Specified as: a fullword integer; where:

- If  $lwork = 0$ , PDGTSV and PDDTSV dynamically allocate the work area used by the subroutine. The work area is deallocated before control is returned to the calling program. This option is an extension to the ScaLAPACK standard.
- If  $lwork = -1$ , PDGTSV and PDDTSV perform a work area query and return the optimum size of  $work$  in  $work_1$ . No computation is performed and the subroutine returns after error checking is complete.
- Otherwise, if (the process grid is  $p \times 1$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 502$ :
  - If  $nrhs \leq 1$ , then:
    - For PDGTSV,  $lwork \geq 18P + MB\_A + 12$ .
    - For PDDTSV,  $lwork \geq 10P + 10$
  - If  $nrhs > 1$ , then:
    - For PDGTSV,  $lwork \geq 24P + 5(MB\_A + nrhs)$
    - For PDDTSV,  $lwork \geq 20P + 2(MB\_A) + 4(nrhs)$

where, in the above formulas,  $P$  is the **actual** number of processes containing data.

If (the process grid is  $1 \times p$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 501$ , you would substitute  $NB\_A$  in place of  $MB\_A$  in the formulas above.

**Note:** In ScaLAPACK 1.5, PDDTSV requires  $lwork = 22P + 3MB\_A + 4(nrhs)$ . This value is greater than or equal to the value required by Parallel ESSL.

*info* See On Return.

### On Return

*dl* is overwritten; that is, the original input is not preserved.

*d* is overwritten; that is, the original input is not preserved.

*du* is overwritten; that is, the original input is not preserved.

*b*  $b$  is the updated local part of the global matrix  $B$ , containing the solution vectors.

Scope: **local**

Returned as: an  $LLD\_B$  by (at least)  $nrhs$  array, containing numbers of the data type indicated in Table 85 on page 518.

*work* is the work area used by this subroutine if  $lwork \neq 0$ , where:

If  $lwork \neq 0$  and  $lwork \neq -1$ , its size is (at least) of length  $lwork$ .

If  $lwork = -1$ , its size is (at least) of length 1.

Scope: **local**

Returned as: an area of storage, containing numbers of the data type indicated in Table 85 on page 518, where:

- If  $lwork \geq 1$ ,  $work_1$  is set to the minimum  $lwork$  value needed.
- If  $lwork = -1$ ,  $work_1$  is set to the optimum  $lwork$  value needed.

Except for  $work_1$ , the contents of  $work$  are overwritten on return.

*info* has the following meaning:

If  $info = 0$ , the factorization or the work area query completed successfully.

**Note:** For PDDTSV, if the input matrix  $A$  is not diagonally dominant, the subroutine may still complete the factorization; however, results are unpredictable.

If  $1 \leq \text{info} \leq p$ , the portion of the global submatrix  $A$  stored on process  $\text{info}-1$  and factored locally, is singular or reducible (for PDGTSV), or not diagonally dominant (for PDDTSV). The magnitude of a pivot element was zero or too small.

If  $\text{info} > p$ , the portion of the global submatrix  $A$  stored on process  $\text{info}-p-1$  representing interactions with other processes, is singular or reducible (for PDGTSV), or not diagonally dominant (for PDDTSV). The magnitude of a pivot element was zero or too small.

If  $\text{info} > 0$ , the results are unpredictable.

Scope: **global**

Returned as: a fullword integer;  $\text{info} \geq 0$ .

## Notes and Coding Rules

1. In your C program, argument *info* must be passed by reference.
2. If  $n > 0$  and  $\text{nrhs} = 0$ , only the factorization is completed.
3. *dl*, *d*, *du*, *B*, and *work* must have no common elements; otherwise, results are unpredictable.
4. In all cases, follow these rules:
  - $ia = ib$
  - $\text{CTXT\_A} = \text{CTXT\_B}$
  - If (the process grid is  $p \times 1$  and  $\text{DTYPE\_A} = 1$ ) or  $\text{DTYPE\_A} = 502$ ,  $\text{MB\_A} = \text{MB\_B}$ .
  - If (the process grid is  $1 \times p$  and  $\text{DTYPE\_A} = 1$ ) or  $\text{DTYPE\_A} = 501$ ,  $\text{NB\_A} = \text{MB\_B}$ .
  - If  $\text{DTYPE\_A} = 1$ , then:
    - For a  $p \times 1$  process grid (where  $p > 1$ ),  $\text{N\_A} = 1$ ,  $\text{NB\_A} \geq 1$ , and  $\text{CSRC\_A} = 0$ .
    - For a  $1 \times p$  process grid,  $\text{M\_A} = 1$ ,  $\text{MB\_A} \geq 1$ , and  $\text{RSRC\_A} = 0$ .
    - For a  $1 \times 1$  process grid:
      - If  $\text{N\_A} = 1$ ,  $\text{NB\_A} \geq 1$ , and  $\text{CSRC\_A} = 0$ .
      - If  $\text{M\_A} = 1$ ,  $\text{MB\_A} \geq 1$ , and  $\text{RSRC\_A} = 0$ .
  - If  $\text{DTYPE\_B} = 1$ ,  $\text{N\_B} \geq \text{nrhs}$ ,  $\text{NB\_B} \geq 1$ , and  $\text{CSRC\_B} = 0$ .
  - Following are the consistent combinations of array descriptor types and process grids, where  $p$  is the number of processes in the process grid:

DTYPE_A	DTYPE_B	Process Grid
501	502	$p \times 1$ or $1 \times p$
502	502	$p \times 1$ or $1 \times p$
501	1	$p \times 1$
502	1	$p \times 1$
1	502	$1 \times p$
1	1	$1 \times 1$

5. To determine the values of  $\text{LOCp}(n)$  used in the argument descriptions, see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28

page 28 for descriptor type-1 or “Determining the Number of Rows or Columns in Your Local Arrays” on page 32 for descriptor type-501 and type-502.

6. For PDGTSV, the global general tridiagonal matrix  $A$  must be non-singular and irreducible. For PDDTSV, the global general tridiagonal matrix  $A$  must be diagonally dominant to ensure numerical accuracy, because no pivoting is performed. These subroutines use the *info* argument to provide information about  $A$ , like ScaLAPACK. However, these subroutines also issue an error message, which differs from ScaLAPACK.
7. The global general tridiagonal matrix  $A$  must be stored in tridiagonal storage mode and distributed over a one-dimensional process grid, using block-cyclic data distribution. See the section on block-cyclically distributing a tridiagonal matrix in “Matrices” on page 40.  
For more information on using block-cyclic data distribution, see “Specifying Block-Cyclically-Distributed Matrices for the Banded Linear Algebraic Equations” on page 29.
8. Matrix  $B$  must be distributed over a one-dimensional process grid, using block-cyclic data distribution. For more information using block-cyclic data distribution, see “Specifying Block-Cyclically-Distributed Matrices for the Banded Linear Algebraic Equations” on page 29. Also, see the section on distributing the right-hand side matrix in “Matrices” on page 40.
9. If  $lwork = -1$  on any process, it must equal  $-1$  on all processes. That is, if a subset of the processes specifies  $-1$  for the work area size, they must all specify  $-1$ .
10. Although global matrices  $A$  and  $B$  may be block-cyclically distributed on a  $1 \times p$  or  $p \times 1$  process grid, the values of  $n$ ,  $ia$ ,  $ib$ ,  $MB\_A$  (if (the process grid is  $p \times 1$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 502$ ),  $NB\_A$  (if (the process grid is  $1 \times p$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 501$ ), must be chosen so that each process has at most one full or partial block of each of the global submatrices  $A$  and  $B$ .
11. For global tridiagonal matrix  $A$ , use of the type-1 array descriptor with a  $p \times 1$  process grid is an extension to ScaLAPACK 1.5. If your application needs to run with both Parallel ESSL and ScaLAPACK 1.5, it is suggested that you use either a type-501 or a type-502 array descriptor for the matrix  $A$ .

## Error Conditions

### Computational Errors

Matrix  $A$  is a singular or reducible matrix (for PDGTSV), or not diagonally dominant (for PDDTSV). For details, see the description of the *info* argument.

### Resource Errors

Unable to allocate workspace

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1.  $DTYPE\_A$  is invalid.
2.  $DTYPE\_B$  is invalid.

#### Stage 2:

1.  $CTXT\_A$  is invalid.

#### Stage 3:

1. This subroutine was called from outside the process grid.



**Stage 4:****Note:** In the following error conditions:

- If  $M\_A = 1$  and  $DTYPE\_A = 1$ , a  $1 \times 1$  process grid is treated as a  $1 \times p$  process grid.
  - If  $N\_A = 1$  and  $DTYPE\_A = 1$ , a  $1 \times 1$  process grid is treated as a  $p \times 1$  process grid.
1. The process grid is not  $1 \times p$  or  $p \times 1$ .
  2.  $CTXT\_A \neq CTXT\_B$
  3.  $n < 0$
  4.  $ia < 1$
  5.  $DTYPE\_A = 1$  and  $M\_A \neq 1$  and  $N\_A \neq 1$   
If (the process grid is  $1 \times p$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 501$ :
  6.  $N\_A < 0$  and  $(n = 0)$ ;  $N\_A < 1$  otherwise
  7.  $NB\_A < 1$
  8.  $n > (NB\_A)(p) - \text{mod}(ia-1, NB\_A)$
  9.  $ia > N\_A$  and  $(n > 0)$
  10.  $ia+n-1 > N\_A$  and  $(n > 0)$
  11.  $CSRC\_A < 0$  or  $CSRC\_A \geq p$
  12.  $NB\_A \neq MB\_B$
  13.  $CSRC\_A \neq RSRC\_B$   
If the process grid is  $1 \times p$  and  $DTYPE\_A = 1$ :
  14.  $M\_A \neq 1$
  15.  $MB\_A < 1$
  16.  $RSRC\_A \neq 0$   
If (the process grid is  $p \times 1$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 502$ :
  17.  $M\_A < 0$  and  $(n = 0)$ ;  $M\_A < 1$  otherwise
  18.  $MB\_A < 1$
  19.  $n > (MB\_A)(p) - \text{mod}(ia-1, MB\_A)$
  20.  $ia > M\_A$  and  $(n > 0)$
  21.  $ia+n-1 > M\_A$  and  $(n > 0)$
  22.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
  23.  $MB\_A \neq MB\_B$
  24.  $RSRC\_A \neq RSRC\_B$   
If the process grid is  $p \times 1$  and  $DTYPE\_A = 1$ :
  25.  $N\_A \neq 1$
  26.  $NB\_A < 1$
  27.  $CSRC\_A \neq 0$   
In all cases:
  28.  $ia \neq ib$
  29.  $DTYPE\_B = 1$  and the process grid is  $1 \times p$  and  $p > 1$
  30.  $nrhs < 0$
  31.  $ib < 1$
  32.  $M\_B < 0$  and  $(n = 0)$ ;  $M\_B < 1$  otherwise
  33.  $MB\_B < 1$
  34.  $ib > M\_B$  and  $(n > 0)$
  35.  $ib+n-1 > M\_B$  and  $(n > 0)$
  36.  $RSRC\_B < 0$  or  $RSRC\_B \geq p$
  37.  $LLD\_B < \max(1, \text{LOCp}(M\_B))$   
If  $DTYPE\_B = 1$ :
  38.  $N\_B < 0$  and  $(nrhs = 0)$ ;  $N\_B < 1$  otherwise
  39.  $N\_B < nrhs$
  40.  $NB\_B < 1$
  41.  $CSRC\_B \neq 0$

In all cases:

42.  $lwork \neq 0$ ,  $lwork \neq -1$ , and  $lwork < (\text{minimum value})$  (For the minimum value, see the  $lwork$  argument description.)

#### Stage 5:

Each of the following global input arguments are checked to determine whether its value is the same on all processes in the process grid:

1.  $n$  differs.
2.  $nrhs$  differs.
3.  $ia$  differs.
4.  $ib$  differs.
5.  $DTYPE\_A$  differs.  
If  $DTYPE\_A = 1$  on all processes:
  6.  $M\_A$  differs.
  7.  $N\_A$  differs.
  8.  $MB\_A$  differs.
  9.  $NB\_A$  differs.
  10.  $RSRC\_A$  differs.
  11.  $CSRC\_A$  differs.
 If  $DTYPE\_A = 501$  on all processes:
  12.  $N\_A$  differs.
  13.  $NB\_A$  differs.
  14.  $CSRC\_A$  differs.
 If  $DTYPE\_A = 502$  on all processes:
  15.  $M\_A$  differs.
  16.  $MB\_A$  differs.
  17.  $RSRC\_A$  differs.
- In all cases:
  18.  $DTYPE\_B$  differs.  
If  $DTYPE\_B = 1$  on all processes:
    19.  $M\_B$  differs.
    20.  $N\_B$  differs.
    21.  $MB\_B$  differs.
    22.  $NB\_B$  differs.
    23.  $RSRC\_B$  differs.
    24.  $CSRC\_B$  differs.
 If  $DTYPE\_B = 502$  on all processes:
    25.  $M\_B$  differs.
    26.  $MB\_B$  differs.
    27.  $RSRC\_B$  differs.
  - Also:
    28.  $lwork = -1$  on a subset of processes.

## Examples

### Example

This example shows a factorization of the general tridiagonal matrix  $A$  of order 12:

$$\begin{bmatrix} 2.0 & 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 3.0 & 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 3.0 & 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 3.0 & 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 3.0 & 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 3.0 & 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 3.0 & 2.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 3.0 & 2.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 3.0 & 2.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 3.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 3.0 & 2.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 3.0 \end{bmatrix}$$

Matrix  $A$  is distributed over a  $1 \times 3$  process grid using block-column distribution.

**Notes:**

1. On output, the vectors  $dl$ ,  $d$ , and  $du$  are overwritten by this subroutine.
2. Notice **only one process grid was created**, even though,  $DTYPE\_A = 501$  and  $DTYPE\_B = 502$ .
3. Because  $lwork = 0$ , this subroutine dynamically allocates the work area used by this subroutine.

**Call Statements and Input:**

```
ORDER = 'R'
NPROW = 1
NPCOL = 3
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      N  NRHS DL  D  DU  IA  DESC_A  B  IB  DESC_B  WORK LWORK INFO
CALL PDGTSV( 12 , 3 , DL , D , DU , 1 , DESC_A , B , 1 , DESC_B , WORK , 0 , INFO )

-or-

      N  NRHS DL  D  DU  IA  DESC_A  B  IB  DESC_B  WORK LWORK INFO
CALL PDDTSV( 12 , 3 , DL , D , DU , 1 , DESC_A , B , 1 , DESC_B , WORK , 0 , INFO )
```

	Desc_A
DTYPE_	501
CTXT_	<i>icontxt</i> <sup>1</sup>
N_	12
NB_	4
CSRC_	0
Not used	—
Reserved	—

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.

	Desc_B
DTYPE_	502
CTXT_	<i>icontxt</i> <sup>1</sup>
M_	12
MB_	4
RSRC_	0
LLD_B	4

## PDGTSV and PDDTSV

	Desc_B
Reserved	—

### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.

Global vector ***dl*** with block size of 4:

$$\begin{array}{c}
 \text{B,D} \qquad \qquad \qquad 0 \qquad \qquad \qquad 1 \qquad \qquad \qquad 2 \\
 0 \left[ \begin{array}{cccc|cccc|cccc}
 . & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0
 \end{array} \right]
 \end{array}$$

Global vector ***d*** with block size of 4:

$$\begin{array}{c}
 \text{B,D} \qquad \qquad \qquad 0 \qquad \qquad \qquad 1 \qquad \qquad \qquad 2 \\
 0 \left[ \begin{array}{cccc|cccc|cccc}
 2.0 & 3.0 & 3.0 & 3.0 & 3.0 & 3.0 & 3.0 & 3.0 & 3.0 & 3.0 & 3.0 & 3.0
 \end{array} \right]
 \end{array}$$

Global vector ***du*** with block size of 4:

$$\begin{array}{c}
 \text{B,D} \qquad \qquad \qquad 0 \qquad \qquad \qquad 1 \qquad \qquad \qquad 2 \\
 0 \left[ \begin{array}{cccc|cccc|cccc}
 2.0 & 2.0 & 2.0 & 2.0 & 2.0 & 2.0 & 2.0 & 2.0 & 2.0 & 2.0 & 2.0 & .
 \end{array} \right]
 \end{array}$$

The following is the  $1 \times 3$  process grid:

$$\begin{array}{c|c|c|c}
 \text{B,D} & 0 & 1 & 2 \\
 \hline
 0 & P_{00} & P_{01} & P_{02}
 \end{array}$$

Local array DL with block size of 4:

$$\begin{array}{c|c|c|c}
 \text{p,q} & 0 & 1 & 2 \\
 \hline
 0 & . & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0
 \end{array}$$

Local array D with block size of 4:

$$\begin{array}{c|c|c|c}
 \text{p,q} & 0 & 1 & 2 \\
 \hline
 0 & 2.0 & 3.0 & 3.0 & 3.0 & 3.0 & 3.0 & 3.0 & 3.0 & 3.0 & 3.0 & 3.0
 \end{array}$$

Local array DU with block size of 4:

$$\begin{array}{c|c|c|c}
 \text{p,q} & 0 & 1 & 2 \\
 \hline
 0 & 2.0 & 2.0 & 2.0 & 2.0 & 2.0 & 2.0 & 2.0 & 2.0 & 2.0 & 2.0 & .
 \end{array}$$

Global matrix ***B*** with a block size of 4:

$$\begin{array}{c}
 \text{B,D} \qquad \qquad \qquad 0 \\
 0 \left[ \begin{array}{ccc}
 46.0 & 6.0 & 4.0 \\
 65.0 & 13.0 & 6.0 \\
 59.0 & 19.0 & 6.0 \\
 53.0 & 25.0 & 6.0 \\
 \hline
 47.0 & 31.0 & 6.0 \\
 41.0 & 37.0 & 6.0
 \end{array} \right]
 \end{array}$$

1		35.0	43.0	6.0	
		29.0	49.0	6.0	
2		-----			
		23.0	55.0	6.0	
		17.0	61.0	6.0	
		11.0	67.0	6.0	
		5.0	47.0	4.0	

The following is the  $1 \times 3$  process grid:

B,D		0		1		2	
-----		-----		-----		-----	
0		P <sub>00</sub>		P <sub>01</sub>		P <sub>02</sub>	

Local matrix  $B$  with a block size of 4:

p,q	0			1			2		
0	46.0	6.0	4.0	47.0	31.0	6.0	23.0	55.0	6.0
	65.0	13.0	6.0	41.0	37.0	6.0	17.0	61.0	6.0
	59.0	19.0	6.0	35.0	43.0	6.0	11.0	67.0	6.0
	53.0	25.0	6.0	29.0	49.0	6.0	5.0	47.0	4.0

**Output:**

Global matrix  $B$  with a block size of 4:

p,q	0		
0	12.0	1.0	1.0
	11.0	2.0	1.0
	10.0	3.0	1.0
	9.0	4.0	1.0
1	8.0	5.0	1.0
	7.0	6.0	1.0
	6.0	7.0	1.0
	5.0	8.0	1.0
2	4.0	9.0	1.0
	3.0	10.0	1.0
	2.0	11.0	1.0
	1.0	12.0	1.0

The following is the  $1 \times 3$  process grid:

B,D		0		1		2	
-----		-----		-----		-----	
0		P <sub>00</sub>		P <sub>01</sub>		P <sub>02</sub>	

Local matrix  $B$  with a block size of 4:

p,q	0			1			2		
0	12.0	1.0	1.0	8.0	5.0	1.0	4.0	9.0	1.0
	11.0	2.0	1.0	7.0	6.0	1.0	3.0	10.0	1.0
	10.0	3.0	1.0	6.0	7.0	1.0	2.0	11.0	1.0
	9.0	4.0	1.0	5.0	8.0	1.0	1.0	12.0	1.0

The value of *info* is 0 on all processes.

## PDGTTRF and PDDTTRF — General Tridiagonal Matrix Factorization

### Purpose

PDGTTRF factors the general tridiagonal matrix  $A$ , stored in tridiagonal storage mode, using Gaussian elimination with partial pivoting.

PDDTTRF factors the diagonally dominant general tridiagonal matrix  $A$ , stored in tridiagonal storage mode, using Gaussian elimination.

In these subroutine descriptions,  $A$  represents the global square general tridiagonal submatrix  $A_{ia:ia+n-1, ia:ia+n-1}$ .

To solve a tridiagonal system of linear equations with multiple right-hand sides, follow the call to PDGTTRF or PDDTTRF with one or more calls to PDGTTRS or PDDTTRS, respectively. The output from these factorization subroutines should be used only as input to the solve subroutines PDGTTRS and PDDTTRS, respectively.

If  $n = 0$ , no computation is performed and the subroutine returns after doing some parameter checking. See reference [54].

Table 90. Data Types

$dl, d, du, du2, af, work$	$ipiv$	Subroutine
Long-precision real	Integer	PDGTTRF and PDDTTRF

### Syntax

<b>Fortran</b>	CALL PDGTTRF ( $n, dl, d, du, du2, ia, desc\_a, ipiv, af, laf, work, lwork, info$ ) CALL PDDTTRF ( $n, dl, d, du, ia, desc\_a, af, laf, work, lwork, info$ )
<b>C and C++</b>	pdgttrf ( $n, dl, d, du, du2, ia, desc\_a, ipiv, af, laf, work, lwork, info$ ); pddttrf ( $n, dl, d, du, ia, desc\_a, af, laf, work, lwork, info$ );

### On Entry

$n$  is the order of the general tridiagonal matrix  $A$  and the number of elements in vector  $ipiv$  used in the computation.

Scope: **global**

Specified as: a fullword integer, where:

- If (the process grid is  $p \times 1$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 502$ ,  $0 \leq n \leq (MB\_A)(p) - \text{mod}(ia-1, MB\_A)$ .
- If (the process grid is  $1 \times p$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 501$ ,  $0 \leq n \leq (NB\_A)(p) - \text{mod}(ia-1, NB\_A)$ .

where  $p$  is the number of processes in a process grid.

$dl$  is the local part of the global vector  $dl$ . This identifies the **first element** of the local array DL. These subroutines compute the location of the first element of the local subarray used, based on  $ia, desc\_a$ , and  $p$ ; therefore, the leading  $LOCp(ia+n-1)$  part of the local array DL contains the local pieces of the leading  $ia+n-1$  part of the global vector.

The global vector  $dl$  contains the subdiagonal of the global general tridiagonal submatrix  $A$  in elements  $ia+1$  through  $ia+n-1$ .

Scope: **local**

Specified as: a one-dimensional array of (at least) length  $\text{LOCp}(ia+n-1)$ , containing numbers of the data type indicated in Table 90 on page 532. Details about block-cyclic data distribution of global matrix  $A$  are stored in  $desc\_a$ .

On output,  $DL$  is overwritten; that is, the original input is not preserved.

$d$  is the local part of the global vector  $d$ . This identifies the **first element** of the local array  $D$ . These subroutines compute the location of the first element of the local subarray used, based on  $ia$ ,  $desc\_a$ , and  $p$ ; therefore, the leading  $\text{LOCp}(ia+n-1)$  part of the local array  $D$  contains the local pieces of the leading  $ia+n-1$  part of the global vector.

The global vector  $d$  contains the main diagonal of the global general tridiagonal submatrix  $A$  in elements  $ia$  through  $ia+n-1$ .

Scope: **local**

Specified as: a one-dimensional array of (at least) length  $\text{LOCp}(ia+n-1)$ , containing numbers of the data type indicated in Table 90 on page 532. Details about block-cyclic data distribution of global matrix  $A$  are stored in  $desc\_a$ .

On output,  $D$  is overwritten; that is, the original input is not preserved.

$du$  is the local part of the global vector  $du$ . This identifies the **first element** of the local array  $DU$ . These subroutines compute the location of the first element of the local subarray used, based on  $ia$ ,  $desc\_a$ , and  $p$ ; therefore, the leading  $\text{LOCp}(ia+n-1)$  part of the local array  $DU$  contains the local pieces of the leading  $ia+n-1$  part of the global vector.

The global vector  $du$  contains the superdiagonal of the global general tridiagonal submatrix  $A$  in elements  $ia$  through  $ia+n-2$ .

Scope: **local**

Specified as: a one-dimensional array of (at least) length  $\text{LOCp}(ia+n-1)$ , containing numbers of the data type indicated in Table 90 on page 532. Details about block-cyclic data distribution of global matrix  $A$  are stored in  $desc\_a$ .

On output,  $DU$  is overwritten; that is, the original input is not preserved.

$du2$  See On Return.

$ia$  is the row or column index of the global matrix  $A$ , identifying the first row or column of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer, where:

- If (the process grid is  $p \times 1$  and  $\text{DTYPE\_A} = 1$ ) or  $\text{DTYPE\_A} = 502$ ,  
 $1 \leq ia \leq M\_A$  and  $ia+n-1 \leq M\_A$
- If (the process grid is  $1 \times p$  and  $\text{DTYPE\_A} = 1$ ) or  $\text{DTYPE\_A} = 501$ ,  
 $1 \leq ia \leq N\_A$  and  $ia+n-1 \leq N\_A$

$desc\_a$  is the array descriptor for global matrix  $A$ . Because vectors are one-dimensional data structures, you may use a type-502, type-501, or type-1 array descriptor regardless of whether the process grid is  $p \times 1$  or  $1 \times p$ . For a type-502 array descriptor, the process grid is used as if it is a  $p \times 1$  process grid. For a type-501 array descriptor, the process grid is used as if it is a  $1 \times p$  process grid. For a type-1 array descriptor, the process

grid is used as if it is either a  $p \times 1$  process grid or a  $1 \times p$  process grid. The following tables describe three types of array descriptors. For rules on using array descriptors, see “Notes and Coding Rules” on page 538.

Table 91. Type-502 Array Descriptor

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor Type	DTYPE_A=502 for $p \times 1$ or $1 \times p$  where $p$ is the number of processes in a process grid.	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	MB_A	Row block size	$MB\_A \geq 1$ and $0 \leq n \leq (MB\_A)(p) - \text{mod}(ia-1, MB\_A)$	Global
5	RSRC_A	The process row over which the first row of the global matrix is distributed	$0 \leq \text{RSRC\_A} < p$	Global
6	—	Not used by these subroutines.	—	—
7	—	Reserved	—	—

Specified as: an array of (at least) length 7, containing fullword integers.

Table 92. Type-1 Array Descriptor ( $p \times 1$  Process Grid)

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor Type	DTYPE_A = 1 for $p \times 1$  where $p$ is the number of processes in a process grid.	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	$N\_A = 1$	
5	MB_A	Row block size	$MB\_A \geq 1$ and $0 \leq n \leq (MB\_A)(p) - \text{mod}(ia-1, MB\_A)$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global
7	RSRC_A	The process row over which the first row of the global matrix is distributed	$0 \leq \text{RSRC\_A} < p$	Global
8	CSRC_A	The process column over which the first column of the global matrix is distributed	$\text{CSRC\_A} = 0$	Global



Table 92. Type-1 Array Descriptor ( $p \times 1$  Process Grid) (continued)

<i>desc_a</i>	Name	Description	Limits	Scope
9	—	Not used by these subroutines.	—	—

Specified as: an array of (at least) length 9, containing fullword integers.

Table 93. Type-501 Array Descriptor

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor Type	DTYPE_A=501 for $1 \times p$ or $p \times 1$ where $p$ is the number of processes in a process grid.	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	N_A	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
4	NB_A	Column block size	$NB\_A \geq 1$ and $0 \leq n \leq (NB\_A)(p) - \text{mod}(ia-1, NB\_A)$	Global
5	CSRC_A	The process column over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < p$	Global
6	—	Not used by these subroutines.	—	—
7	—	Reserved	—	—

Specified as: an array of (at least) length 7, containing fullword integers.

Table 94. Type-1 Array Descriptor ( $1 \times p$  Process Grid)

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A = 1 for $1 \times p$ where $p$ is the number of processes in a process grid.	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	$M\_A = 1$	Global
4	N_A	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$ and $0 \leq n \leq (NB\_A)(p) - \text{mod}(ia-1, NB\_A)$	Global

## PDGTTRF and PDDTTRF

Table 94. Type-1 Array Descriptor ( $1 \times p$  Process Grid) (continued)

<i>desc_a</i>	Name	Description	Limits	Scope
7	RSRC_A	The process row over which the first row of the global matrix is distributed	RSRC_A = 0	Global
8	CSRC_A	The process column over which the first column of the global matrix is distributed	$0 \leq \text{CSRC\_A} < p$	Global
9	—	Not used by these subroutines.	—	—

Specified as: an array of (at least) length 9, containing fullword integers.

*ipiv* See On Return.

*af* See On Return.

*laf* is the number of elements in array AF.

Scope: **local**

Specified as: a fullword integer, where:

If (the process grid is  $p \times 1$  and DTYPE\_A = 1) or DTYPE\_A = 502:

- For PDGTTRF,  $laf \geq 12P+3(\text{MB\_A})$
- For PDDTTRF,  $laf \geq 12P+2(\text{MB\_A})$ .

where, in the above formulas, P is the **actual** number of processes containing data.

If (the process grid is  $1 \times p$  and DTYPE\_A = 1) or DTYPE\_A = 501, you would substitute NB\_A in place of MB\_A in the formulas above.

**Note:** In ScaLAPACK 1.5, PDDTTRF requires  $laf = 12P+3\text{NB\_A}$ . This value is greater than or equal to the value required by Parallel ESSL.

*work* has the following meaning:

If  $lwork = 0$ , *work* is ignored.

If  $lwork \neq 0$ , *work* is the work area used by this subroutine, where:

- If  $lwork \neq -1$ , the size of *work* is (at least) of length *lwork*.
- If  $lwork = -1$ , the size of *work* is (at least) of length 1.

Scope: **local**

Specified as: an area of storage containing numbers of data type indicated in Table 90 on page 532.

*lwork* is the number of elements in array WORK.

Scope:

- If  $lwork \geq 0$ , *lwork* is **local**
- If  $lwork = -1$ , *lwork* is **global**

Specified as: a fullword integer; where:

- If  $lwork = 0$ , PDGTTRF and PDDTTRF dynamically allocate the work area used by the subroutine. The work area is deallocated before control is returned to the calling program. This option is an extension to the ScaLAPACK standard.

- If  $lwork = -1$ , PDGTTTRF and PDDTTTRF perform a work area query and return the optimum size of  $work$  in  $work_1$ . No computation is performed and the subroutine returns after error checking is complete.
- Otherwise,  $lwork$  must have the following value:  
 For PDGTTTRF,  $lwork \geq 10P$   
 For PDDTTTRF,  $lwork \geq 8P$   
 where, in the above formulas,  $P$  is the **actual** number of processes containing data.

*info* See On Return.

### On Return

*dl* *dl* is the updated local part of the global vector *dl*, containing part of the factorization.

Scope: **local**

Returned as: a one-dimensional array of (at least)  $LOCp(ia+n-1)$ , containing numbers of the data type indicated in Table 90 on page 532.

On output, *DL* is overwritten; that is, the original input is not preserved.

*d* *d* is the updated local part of the global vector *d*, containing part of the factorization.

Scope: **local**

Returned as: a one-dimensional array of (at least) length  $LOCp(ia+n-1)$ , containing numbers of the data type indicated in Table 90 on page 532.

On output, *D* is overwritten; that is, the original input is not preserved.

*du* *du* is the updated local part of the global vector *du*, containing part of the factorization.

Scope: **local**

Returned as: a one-dimensional array of (at least) length  $LOCp(ia+n-1)$ , containing numbers of the data type indicated in Table 90 on page 532.

On output, *DU* is overwritten; that is, the original input is not preserved.

*du2* is the local part of the global vector *du2*, containing part of the factorization.

Scope: **local**

Returned as: a one-dimensional array of (at least) length  $LOCp(ia+n-1)$ , containing numbers of the data type indicated in Table 90 on page 532.

*ipiv* is the local part of the global vector *ipiv*, containing the pivot information needed by PDGTTTRS. This identifies the **first element** of the local array IPIV. These subroutines compute the location of the first element of the local subarray used, based on *ia*, *desc\_a*, and *p*; therefore, the leading  $LOCp(ia+n-1)$  part of the local array IPIV contains the local pieces of the leading  $ia+n-1$  part of the global vector.

Scope: **local**

Returned as: an array of (at least) length  $LOCp(ia+n-1)$ , containing fullword integers. There is no array descriptor for *ipiv*. The details about the block data distribution of global vector *ipiv* are stored in *desc\_a*.

*af* is a work area used by these subroutines and contains part of the factorization. Its size is specified by *laf*.

Scope: **local**

Returned as: a one-dimensional array of (at least) length *laf*, containing numbers of the data type indicated in Table 90 on page 532.

*work* is the work area used by this subroutine if *lwork*  $\neq$  0, where:

If *lwork*  $\neq$  0 and *lwork*  $\neq$  -1, the size of *work* is (at least) of length *lwork*.

If *lwork* = -1, the size of *work* is (at least) of length 1.

Scope: **local**

Returned as: an area of storage, containing numbers of data type indicated in Table 90 on page 532, where:

- If *lwork*  $\geq$  1, the *work*<sub>1</sub> is set to the minimum *lwork* value needed.
- If *lwork* = -1, the *work*<sub>1</sub> is set to the optimum *lwork* value needed.

Except for *work*<sub>1</sub>, the contents of *work* are overwritten on return.

*info* has the following meaning:

If *info* = 0, the factorization or work area query completed successfully.

**Note:** For PDDTTRF, if the input matrix *A* is not diagonally dominant, the subroutine may still complete the factorization; however, results are unpredictable.

If  $1 \leq \textit{info} \leq p$ , the portion of the global submatrix *A* stored on process *info*-1 and factored locally, is singular or reducible (for PDGTTRF), or not diagonally dominant (for PDDTTRF). The magnitude of a pivot element was zero or too small.

If *info* > *p*, the portion of the global submatrix *A* stored on process *info*-*p*-1 representing interactions with other processes, is singular or reducible (for PDGTTRF), or not diagonally dominant (for PDDTTRF). The magnitude of a pivot element was zero or too small.

If *info* > 0, the factorization is completed; however, if you call PDGTTRS/PDDTTRS with these factors, results are unpredictable.

Scope: **global**

Returned as: a fullword integer; *info*  $\geq$  0.

## Notes and Coding Rules

1. In your C program, argument *info* must be passed by reference.
2. The output from these factorization subroutines should be used only as input to the solve subroutines PDGTTRS and PDDTTRS, respectively.

The factored matrix *A* is stored in an internal format that depends on the number of processes.

The format of the output from PDDTTRF has changed. Therefore, the factorization and solve must be performed using Parallel ESSL Version 2 Release 1.2, or later.

The scalar data specified for input argument *n* must be the same for both PDGTTRF/PDDTTRF and PDGTTRS/PDDTTRS.

The global vectors for *dl*, *d*, *du*, *du2*, and *af* input to PDGTTTRS/PDDTTTRS must be the same as the corresponding output arguments for PDGTTTRF/PDDTTTRF; and thus, the scalar data specified for *ia*, *desc\_a*, and *laf* must also be the same.

3. In all cases, follow these rules:

- $ia = ib$
- If DTYPE\_A=1, then:
  - For a  $p \times 1$  process grid (where  $p > 1$ ), N\_A=1, NB\_A $\geq$ 1, and CSRC\_A=0.
  - For a  $1 \times p$  process grid (where  $p > 1$ ), M\_A=1, MB\_A $\geq$ 1, and RSRC\_A=0.
  - For a  $1 \times 1$  process grid:
    - If N\_A=1, NB\_A $\geq$ 1 and CSRC\_A=0.
    - If M\_A=1, MB\_A $\geq$ 1 and RSRC\_A=0.
- Following are the consistent combinations of array descriptor types and process grids, where  $p$  is the number of processes in the process grid:

DTYPE_A	Process Grid
501	$p \times 1$ or $1 \times p$
502	$p \times 1$ or $1 \times p$
1	$p \times 1$ or $1 \times p$

4. To determine the values of LOCp( $n$ ) used in the argument descriptions, see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28 for descriptor type-1 or “Determining the Number of Rows or Columns in Your Local Arrays” on page 32 for descriptor type-501 and type-502.

5. *dl*, *d*, *du*, *du2*, *ipiv*, *af*, and *work* must have no common elements; otherwise, results are unpredictable.

6. For PDGTTTRF, the global general tridiagonal matrix *A* must be non-singular and irreducible. For PDDTTTRF, the global general tridiagonal matrix *A* must be diagonally dominant to ensure numerical accuracy, because no pivoting is performed. These subroutines use the *info* argument to provide information about *A*, like ScaLAPACK. However, these subroutines also issue an error message, which differs from ScaLAPACK.

7. The global general tridiagonal matrix *A* must be stored in tridiagonal storage mode and distributed over a one-dimensional process grid, using block-cyclic data distribution. See the section on block-cyclically distributing a tridiagonal matrix in “Matrices” on page 40.

For more information on using block-cyclic data distribution, see “Specifying Block-Cyclically-Distributed Matrices for the Banded Linear Algebraic Equations” on page 29.

8. If *lwork* = -1 on any process, it must equal -1 on all processes. That is, if a subset of the processes specifies -1 for the work area size, they must all specify -1.

9. Although global matrix *A* may be block-cyclically distributed on a  $1 \times p$  or  $p \times 1$  process grid, the values of *n*, *ia*, MB\_A (if the process grid is  $p \times 1$  and DTYPE\_A = 1) or DTYPE\_A = 502), NB\_A (if the process grid is  $1 \times p$  and DTYPE\_A = 1) or DTYPE\_A = 501), must be chosen so that each process has at most one full or partial block of global submatrix *A*.

10. For global tridiagonal matrix *A*, use of the type-1 array descriptor is an extension to ScaLAPACK 1.5. If your application needs to run with both Parallel ESSL and ScaLAPACK 1.5, it is suggested that you use either a type-501 or a type-502 array descriptor for the matrix *A*.

## Error Conditions

### Computational Errors

Matrix  $A$  is a singular or reducible matrix (for PDGTTTRF), or not diagonally dominant (for PDDTTTRF). For details, see the description of the *info* argument.

### Resource Errors

Unable to allocate workspace

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. DTYPE\_A is invalid.

#### Stage 2:

1. CTXT\_A is invalid.

#### Stage 3:

1. This subroutine was called from outside the process grid.

#### Stage 4:

**Note:** In the following error conditions:

- If  $M\_A = 1$  and  $DTYPE\_A = 1$ , a  $1 \times 1$  process grid is treated as a  $1 \times p$  process grid.
  - If  $N\_A = 1$  and  $DTYPE\_A = 1$ , a  $1 \times 1$  process grid is treated as a  $p \times 1$  process grid.
1. The process grid is not  $1 \times p$  or  $p \times 1$ .
  2.  $n < 0$
  3.  $ia < 1$
  4.  $DTYPE\_A = 1$  and  $M\_A \neq 1$  and  $N\_A \neq 1$   
If (the process grid is  $1 \times p$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 501$ :
  5.  $N\_A < 0$  and ( $n = 0$ );  $N\_A < 1$  otherwise
  6.  $NB\_A < 1$
  7.  $n > (NB\_A)(p) - \text{mod}(ia-1, NB\_A)$
  8.  $ia > N\_A$  and ( $n > 0$ )
  9.  $ia+n-1 > N\_A$  and ( $n > 0$ )
  10.  $CSRC\_A < 0$  or  $CSRC\_A \geq p$   
If the process grid is  $1 \times p$  and  $DTYPE\_A = 1$ :
  11.  $M\_A \neq 1$
  12.  $MB\_A < 1$
  13.  $RSRC\_A \neq 0$   
If (the process grid is  $p \times 1$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 502$ :
  14.  $M\_A < 0$  and ( $n = 0$ );  $M\_A < 1$  otherwise
  15.  $MB\_A < 1$
  16.  $n > (MB\_A)(p) - \text{mod}(ia-1, MB\_A)$
  17.  $ia > M\_A$  and ( $n > 0$ )
  18.  $ia+n-1 > M\_A$  and ( $n > 0$ )
  19.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$   
If the process grid is  $p \times 1$  and  $DTYPE\_A = 1$ :
  20.  $N\_A \neq 1$
  21.  $NB\_A < 1$
  22.  $CSRC\_A \neq 0$

In all cases:

23.  $laf < (\text{minimum value})$  (For the minimum value, see the  $laf$  argument description.)
24.  $lwork \neq 0$ ,  $lwork \neq -1$ , and  $lwork < (\text{minimum value})$  (For the minimum value, see the  $lwork$  argument description.)

#### Stage 5:

Each of the following global input arguments are checked to determine whether its value is the same on all processes in the process grid:

1.  $n$  differs.
2.  $ia$  differs.
3.  $DTYPE\_A$  differs.  
If  $DTYPE\_A = 1$  on all processes:
  4.  $M\_A$  differs.
  5.  $N\_A$  differs.
  6.  $MB\_A$  differs.
  7.  $NB\_A$  differs.
  8.  $RSRC\_A$  differs.
  9.  $CSRC\_A$  differs.
 If  $DTYPE\_A = 501$  on all processes:
  10.  $N\_A$  differs.
  11.  $NB\_A$  differs.
  12.  $CSRC\_A$  differs.
 If  $DTYPE\_A = 502$  on all processes:
  13.  $M\_A$  differs.
  14.  $MB\_A$  differs.
  15.  $RSRC\_A$  differs.
- Also:
  16.  $lwork = -1$  on a subset of processes.

## Examples

### Example 1

This example shows a factorization of the general tridiagonal matrix  $A$  of order 12.

$$\begin{bmatrix} 2.0 & 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 3.0 & 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 3.0 & 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 3.0 & 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 3.0 & 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 3.0 & 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 3.0 & 2.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 3.0 & 2.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 3.0 & 2.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 3.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 3.0 & 2.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 3.0 \end{bmatrix}$$

Matrix  $A$  is stored in tridiagonal storage mode and is distributed over a  $3 \times 1$  process grid using block-cyclic distribution.

#### Notes:

1. The vectors  $dl$ ,  $d$ , and  $du$ , output from PDGTTTRF, are stored in an internal format that depends on the number of processes. These vectors are passed, unchanged, to the solve subroutine PDGTTRS.

## PDGTTRF and PDDTTRF

2. The contents of the *du2* and *af* vectors, output from PDGTTRF, are not shown. These vectors are passed, unchanged, to the solve subroutine PDGTTRS.
3. Because *lwork* = 0, PDGTTRF dynamically allocates the work area used by this subroutine.

### Call Statements and Input:

```
ORDER = 'R'
NPROW = 3
NPCOL = 1
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      N   DL   D   DU   DU2  IA   DESC_A   IPIV   AF   LAF   WORK  LWORK  INFO
CALL PDGTTRF( 12 , DL , D , DU , DU2 , 1 , DESC_A , IPIV , AF , 48 , WORK , 0 , INFO )
```

	Desc_A
DTYPE_	502
CTXT_	<i>icontxt</i> <sup>1</sup>
M_	12
MB_	4
RSRC_	0
Not used	—
Reserved	—

### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.

Global vector *dl* with block size of 4:

```
B,D      0
0  [ 1.0
    1.0
    1.0
    ---
    1.0
    1.0
    1.0
    1.0
    1.0
    1.0
    1.0
    1.0 ]
```

Global vector *d* with block size of 4:

```
B,D      0
0  [ 2.0
    3.0
    3.0
    3.0
    ---
    3.0 ]
```



1	3.0
	3.0
	3.0
	---
2	3.0
	3.0
	3.0
	3.0

Global vector *du* with block size of 4:

B,D	0
0	2.0
	2.0
	2.0
	2.0
1	---
	2.0
	2.0
	2.0
2	---
	2.0
	2.0
	2.0
	.

The following is the 3 × 1 process grid:

B,D	0
0	P <sub>00</sub>
1	P <sub>10</sub>
2	P <sub>20</sub>

Local array DL with block size of 4:

p,q	0
0	.
	1.0
	1.0
	1.0
1	---
	1.0
	1.0
	1.0
2	---
	1.0
	1.0
	1.0

Local array D with block size of 4:

p,q	0
0	2.0
	3.0
	3.0
	3.0

## PDGTTRF and PDDTTRF

1	3.0
	3.0
	3.0
	3.0
2	3.0
	3.0
	3.0
	3.0

Local array DU with block size of 4:

p,q	0
0	2.0
	2.0
	2.0
	2.0
1	2.0
	2.0
	2.0
	2.0
2	2.0
	2.0
	2.0
	.

**Output:**

Global vector  $dl$  with block size of 4:

B,D	0
0	.
	0.5
	0.5
	0.5
1	1.0
	0.33
	0.43
	0.47
2	1.0
	1.0
	1.0
	1.0

Global vector  $d$  with block size of 4:

B,D	0
0	0.5
	0.5
	0.5
	2.0
1	0.33
	0.43
	0.47
	2.07
2	2.07
	2.07
	2.07
	2.07

$$2 \quad \left[ \begin{array}{c} 0.47 \\ 0.43 \\ 0.33 \end{array} \right]$$

Global vector *du* with block size of 4:

$$\begin{array}{cc} \text{B,D} & 0 \\ & \left[ \begin{array}{c} 2.0 \\ 2.0 \\ 2.0 \\ 2.0 \\ \text{----} \\ 2.0 \\ 2.0 \\ 2.0 \\ 2.0 \\ \text{----} \\ 0.93 \\ 0.86 \\ 0.67 \\ . \end{array} \right] \end{array}$$

Global vector *ipiv* with block size of 4:

$$\begin{array}{cc} \text{B,D} & 0 \\ & \left[ \begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ - \\ 0 \\ 0 \\ 0 \\ 0 \\ - \\ 0 \\ 0 \\ 0 \\ 0 \end{array} \right] \end{array}$$

The following is the 3 × 1 process grid:

$$\begin{array}{cc} \text{B,D} & 0 \\ \text{----} & \text{----} \\ 0 & P_{00} \\ \text{----} & \text{----} \\ 1 & P_{10} \\ \text{----} & \text{----} \\ 2 & P_{20} \end{array}$$

Local array DL with block size of 4:

$$\begin{array}{cc} \text{p,q} & 0 \\ \text{----} & \text{----} \\ & . \\ & 0.5 \\ 0 & 0.5 \\ & 0.5 \\ \text{----} & \text{----} \\ & 1.0 \\ & 0.33 \\ 1 & 0.43 \\ & 0.47 \end{array}$$

## PDGTTRF and PDDTTRF

	-----
	1.0
	1.0
2	1.0
	1.0

Local array D with block size of 4:

p,q	0
	-----
	0.5
	0.5
0	0.5
	2.0
	-----
	0.33
	0.43
1	0.47
	2.07
	-----
	2.07
	0.47
2	0.43
	0.33

Local array DU with block size of 4:

p,q	0
	-----
	2.0
	2.0
0	2.0
	2.0
	-----
	2.0
	2.0
1	2.0
	2.0
	-----
	0.93
	0.86
2	0.67
	.

Local array IPIV with block size of 4:

p,q	0
	----
	0
	0
0	0
	0
	----
	0
	0
1	0
	0
	----
	0
	0
2	0
	0

The value of *info* is 0 on all processes.

## Example 2

This example shows a factorization of the diagonally dominant general tridiagonal matrix  $A$  of order 12. Matrix  $A$  is stored in tridiagonal storage mode and distributed over a  $3 \times 1$  process grid using block-cyclic distribution.

Matrix  $A$  and the input and/or output values for  $dl$ ,  $d$ ,  $du$ ,  $desc\_a$ , and  $info$  in this example are the same as shown for “Example 1” on page 541.

### Notes:

1. The vectors  $dl$ ,  $d$ , and  $du$ , output from PDDTTTRF, are stored in an internal format that depends on the number of processes. These vectors are passed, unchanged, to the solve subroutine PDDTTTRS.
2. The contents of vector  $af$ , output from PDDTTTRF, are not shown. This vector is passed, unchanged, to the solve subroutine PDDTTTRS.
3. Because  $lwork = 0$ , PDDTTTRF dynamically allocates the work area used by this subroutine.

### Call Statements and Input:

```
ORDER = 'R'
NPROW = 3
NPCOL = 1
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      N   DL   D   DU   IA  DESC_A  AF   LAF  WORK  LWORK  INFO
      |   |   |   |   |   |   |   |   |   |   |
CALL PDDTTTRF( 12 , DL , D , DU , 1 , DESC_A , AF , 44 , WORK , 0 , INFO )
```

## PDGTTRS and PDDTTRS — General Tridiagonal Matrix Solve

### Purpose

PDGTTRS solves the tridiagonal systems of linear equations, using Gaussian elimination with partial pivoting for the general tridiagonal matrix  $A$  stored in tridiagonal storage mode.

- 1.  $AX = B$

PDDTTRS solves one of the following tridiagonal systems of linear equations, using Gaussian elimination for the diagonally dominant general tridiagonal matrix  $A$  stored in tridiagonal storage mode.

- 1.  $AX = B$
- 2.  $A^T X = B$

In these subroutines:

- $A$  represents the global square general tridiagonal submatrix  $A_{ia:ia+n-1, ia:ia+n-1}$ .
- $B$  represents the global general submatrix  $B_{ib:ib+n-1, 1:nrhs}$  containing the right-hand sides in its columns.
- $X$  represents the global general submatrix  $B_{ib:ib+n-1, 1:nrhs}$  containing the output solution vectors in its columns.

These subroutines use the results of the factorization of matrix  $A$ , produced by a preceding call to PDGTTRF or PDDTTRF, respectively. The output from the factorization subroutines, PDGTTRF and PDDTTRF, should be used only as input to these solve subroutines, respectively.

If  $n = 0$  or  $nrhs = 0$ , no computation is performed and the subroutine returns after doing some parameter checking. See reference [54].

Table 95. Data Types

$dl, d, du, du2, B, af, work$	$ipiv$	Subroutine
Long-precision real	Integer	PDGTTRS and PDDTTRS

### Syntax

Fortran	CALL PDGTTRS ( <i>transa, n, nrhs, dl, d, du, du2, ia, desc_a, ipiv, b, ib, desc_b, af, laf, work, lwork, info</i> )
	CALL PDDTTRS ( <i>transa, n, nrhs, dl, d, du, ia, desc_a, b, ib, desc_b, af, laf, work, lwork, info</i> )
C and C++	pdgttrs ( <i>transa, n, nrhs, dl, d, du, du2, ia, desc_a, ipiv, b, ib, desc_b, af, laf, work, lwork, info</i> );
	pddttrs ( <i>transa, n, nrhs, dl, d, du, ia, desc_a, b, ib, desc_b, af, laf, work, lwork, info</i> );

### On Entry

*transa* indicates submatrix  $A$  is used in the computation, resulting in solution 1.

Scope: **global**

Specified as: a single character, where:

- For PDGTTRS, it must be 'N'.
- For PDDTTRS, it must be 'N', 'T', or 'C'.

*n* is the order of the general tridiagonal submatrix *A* and the number of rows in the general submatrix *B*, which contains the multiple right-hand sides.

Scope: **global**

Specified as: a fullword integer, where:

- If (the process grid is  $p \times 1$  and  $\text{DTYPE\_A} = 1$ ) or  $\text{DTYPE\_A} = 502$ ,  
 $0 \leq n \leq (\text{MB\_A})(p) - \text{mod}(ia-1, \text{MB\_A})$ .
- If (the process grid is  $1 \times p$  and  $\text{DTYPE\_A} = 1$ ) or  $\text{DTYPE\_A} = 501$ ,  
 $0 \leq n \leq (\text{NB\_A})(p) - \text{mod}(ia-1, \text{NB\_A})$ .

where  $p$  is the number of processes in a process grid.

*nrhs* is the number of right-hand sides; that is, the number of columns in submatrix *B* used in the computation.

Scope: **global**

Specified as: a fullword integer;  $nrhs \geq 0$ .

*dl* is the local part of the global vector *dl*, containing part of the factorization produced from a preceding call to PDGTTRF or PDDTTRF. This identifies the **first element** of the local array DL. These subroutines compute the location of the first element of the local subarray used, based on *ia*, *desc\_a*, and *p*; therefore, the leading  $\text{LOCp}(ia+n-1)$  part of the local array DL contains the local pieces of the leading  $ia+n-1$  part of the global vector.

Scope: **local**

Specified as: a one-dimensional array of (at least) length  $\text{LOCp}(ia+n-1)$ , containing numbers of the data type indicated in Table 95 on page 548. Details about block-cyclic data distribution of global matrix *A* are stored in *desc\_a*.

*d* is the local part of the global vector *d*, containing part of the factorization produced from a preceding call to PDGTTRF or PDDTTRF. This identifies the **first element** of the local array D. These subroutines compute the location of the first element of the local subarray used, based on *ia*, *desc\_a*, and *p*; therefore, the leading  $\text{LOCp}(ia+n-1)$  part of the local array D contains the local pieces of the leading  $ia+n-1$  part of the global vector.

Scope: **local**

Specified as: a one-dimensional array of (at least) length  $\text{LOCp}(ia+n-1)$ , containing numbers of the data type indicated in Table 95 on page 548. Details about block-cyclic data distribution of global matrix *A* are stored in *desc\_a*.

*du* is the local part of the global vector *du*, containing part of the factorization produced from a preceding call to PDGTTRF or PDDTTRF. This identifies the **first element** of the local array DU. These subroutines compute the location of the first element of the local subarray used, based on *ia*, *desc\_a*, and *p*; therefore, the leading  $\text{LOCp}(ia+n-1)$  part of the local array DU contains the local pieces of the leading  $ia+n-1$  part of the global vector.

Scope: **local**

Specified as: a one-dimensional array of (at least) length  $\text{LOCp}(ia+n-1)$ , containing numbers of the data type indicated in Table 95 on page 548. Details about block-cyclic data distribution of global matrix *A* are stored in *desc\_a*.

*du2* is the local part of the global vector *du2*, containing part of the

factorization produced from a preceding call to PDGTTRF. This identifies the **first element** of the local array DU2. These subroutines compute the location of the first element of the local subarray used, based on  $ia$ ,  $desc\_a$ , and  $p$ ; therefore, the leading  $LOCp(ia+n-1)$  part of the local array DU2 contains the local pieces of the leading  $ia+n-1$  part of the global vector.

Scope: **local**

Specified as: a one-dimensional array of (at least) length  $LOCp(ia+n-1)$ , containing numbers of the data type indicated in Table 95 on page 548. Details about block-cyclic data distribution of global matrix  $A$  are stored in  $desc\_a$ .

$ia$  is the row or column index of the global matrix  $A$ , identifying the first row or column of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer, where:

- If (the process grid is  $p \times 1$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 502$ ,  $1 \leq ia \leq M\_A$  and  $ia+n-1 \leq M\_A$
- If (the process grid is  $1 \times p$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 501$ ,  $1 \leq ia \leq N\_A$  and  $ia+n-1 \leq N\_A$

$desc\_a$  is the array descriptor for global matrix  $A$ . Because vectors are one-dimensional data structures, you may use a type-502, type-501, or type-1 array descriptor regardless of whether the process grid is  $p \times 1$  or  $1 \times p$ . For a type-502 array descriptor, the process grid is used as if it is a  $p \times 1$  process grid. For a type-501 array descriptor, the process grid is used as if it is a  $1 \times p$  process grid. For a type-1 array descriptor, the process grid is used as if it is either a  $p \times 1$  process grid or a  $1 \times p$  process grid. The following tables describe three types of array descriptors. For rules on using array descriptors, see “Notes and Coding Rules” on page 555.

Table 96. Type-502 Array Descriptor

$desc\_a$	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor Type	DTYPE_A=502 for $p \times 1$ or $1 \times p$ where $p$ is the number of processes in a process grid.	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	MB_A	Row block size	$MB\_A \geq 1$ and $0 \leq n \leq (MB\_A)(p) - \text{mod}(ia-1, MB\_A)$	Global
5	RSRC_A	The process row over which the first row of the global matrix is distributed	$0 \leq RSRC\_A < p$	Global
6	—	Not used by these subroutines.	—	—
7	—	Reserved	—	—

Specified as: an array of (at least) length 7, containing fullword integers.



Table 97. Type-1 Array Descriptor ( $p \times 1$  Process Grid)

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor Type	DTYPE_A = 1 for $p \times 1$ where $p$ is the number of processes in a process grid.	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $n = 0$ : M_A $\geq 0$ Otherwise: M_A $\geq 1$	Global
4	N_A	Number of columns in the global matrix	N_A = 1	
5	MB_A	Row block size	MB_A $\geq 1$ and $0 \leq n \leq (\text{MB\_A})(p) - \text{mod}(ia-1, \text{MB\_A})$	Global
6	NB_A	Column block size	NB_A $\geq 1$	Global
7	RSRC_A	The process row over which the first row of the global matrix is distributed	$0 \leq \text{RSRC\_A} < p$	Global
8	CSRC_A	The process column over which the first column of the global matrix is distributed	CSRC_A = 0	Global
9	—	Not used by these subroutines.	—	—

Specified as: an array of (at least) length 9, containing fullword integers.

Table 98. Type-501 Array Descriptor

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor Type	DTYPE_A=501 for $1 \times p$ or $p \times 1$ where $p$ is the number of processes in a process grid.	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	N_A	Number of columns in the global matrix	If $n = 0$ : N_A $\geq 0$ Otherwise: N_A $\geq 1$	Global
4	NB_A	Column block size	NB_A $\geq 1$ and $0 \leq n \leq (\text{NB\_A})(p) - \text{mod}(ia-1, \text{NB\_A})$	Global
5	CSRC_A	The process column over which the first column of the global matrix is distributed	$0 \leq \text{CSRC\_A} < p$	Global
6	—	Not used by these subroutines.	—	—
7	—	Reserved	—	—

Specified as: an array of (at least) length 7, containing fullword integers.

## PDGTTRS and PDDTTRS

Table 99. Type-1 Array Descriptor ( $1 \times p$  Process Grid)

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A = 1 for $1 \times p$ where $p$ is the number of processes in a process grid.	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	M_A = 1	Global
4	N_A	Number of columns in the global matrix	If $n = 0$ : N_A $\geq 0$ Otherwise: N_A $\geq 1$	Global
5	MB_A	Row block size	MB_A $\geq 1$	Global
6	NB_A	Column block size	NB_A $\geq 1$ and $0 \leq n \leq (NB\_A)(p) - \text{mod}(ia-1, NB\_A)$	Global
7	RSRC_A	The process row over which the first row of the global matrix is distributed	RSRC_A = 0	Global
8	CSRC_A	The process column over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < p$	Global
9	—	Not used by these subroutines.	—	—

Specified as: an array of (at least) length 9, containing fullword integers.

*ipiv* is the local part of the global vector *ipiv*, containing the pivot indices produced on a preceding call to PDGTTRF. This identifies the **first element** of the local array IPIV. This subroutine computes the location of the first element of the local subarray used, based on *ia*, *desc\_a*, and *p*; therefore, the leading LOCp(*ia+n-1*) part of the local array IPIV must contain the local pieces of the leading *ia+n-1* part of the global vector.

Scope: **local**

Specified as: an array of (at least) LOCp(*ia+n-1*), containing fullword integers. There is no array descriptor for *ipiv*. The details about the block-cyclic data distribution of global matrix *A* are stored in *desc\_a*.

*b* is the local part of the global general matrix *B*, containing the multiple right-hand sides of the system. This identifies the **first element** of the local array B. This subroutine computes the location of the first element of the local subarray used, based on *ib*, *desc\_b*, and *p*; therefore, the leading LOCp(*ib+n-1*) by *nrhs* part of the local array B must contain the local pieces of the leading *ib+n-1* by *nrhs* part of the global matrix.

Scope: **local**

Specified as: an LLD\_B by (at least) *nrhs* array, containing numbers of the data type indicated in Table 95 on page 548. Details about the block-cyclic data distribution of global matrix *B* are stored in *desc\_b*.

*ib* is the row index of the global matrix *B*, identifying the first row of the submatrix *B*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ib \leq M\_B$  and  $ib+n-1 \leq M\_B$ .

*desc\_b* is the array descriptor for global matrix *B*, which may be type 502 or type 1, as described in the following tables. For type-502 array descriptor, the process grid is used as if it is a  $p \times 1$  process grid. For rules on using array descriptors, see “Notes and Coding Rules” on page 555.

<i>desc_b</i>	Name	Description	Limits	Scope
1	DTYPE_B	Descriptor type	DTYPE_B = 502 for $p \times 1$ or $1 \times p$ where $p$ is the number of processes in a process grid.	Global
2	CTXT_B	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_B	Number of rows in the global matrix	If $n = 0$ : $M\_B \geq 0$ Otherwise: $M\_B \geq 1$	Global
4	MB_B	Row block size	$MB\_B \geq 1$ and $0 \leq n \leq (MB\_B)p - \text{mod}(ib-1, MB\_B)$	Global
5	RSRC_B	The process row over which the first row of the global matrix is distributed	$0 \leq \text{RSRC\_B} < p$	Global
6	LLD_B	Leading dimension	$LLD\_B \geq \max(1, \text{LOCp}(M\_B))$	Local
7	—	Reserved	—	—

Specified as: an array of (at least) length 7, containing fullword integers.

<i>desc_b</i>	Name	Description	Limits	Scope
1	DTYPE_B	Descriptor type	DTYPE_B = 1 for $p \times 1$ where $p$ is the number of processes in a process grid.	Global
2	CTXT_B	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_B	Number of rows in the global matrix	If $n = 0$ : $M\_B \geq 0$ Otherwise: $M\_B \geq 1$	Global
4	N_B	Number of columns in the global matrix	$N\_B \geq nrhs$	Global
5	MB_B	Row block size	$MB\_B \geq 1$ and $0 \leq n \leq (MB\_B)p - \text{mod}(ib-1, MB\_B)$	Global
6	NB_B	Column block size	$NB\_B \geq 1$	Global
7	RSRC_B	The process row over which the first row of the global matrix is distributed	$0 \leq \text{RSRC\_B} < p$	Global
8	CSRC_B	The process column over which the first column of the global matrix is distributed	$\text{CSRC\_B} = 0$	Global

## PDGTTRS and PDDTTRS

<i>desc_b</i>	Name	Description	Limits	Scope
9	LLD_B	Leading dimension	$LLD\_B \geq \max(1, LOCp(M\_B))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

*af* is a work area used by these subroutines and contains part of the factorization produced on a preceding call to PDGTTRF or PDDTTRF. Its size is specified by *laf*.

Scope: **local**

Specified as: a one-dimensional array of (at least) length *laf*, containing numbers of the data type indicated in Table 95 on page 548.

*laf* is the number of elements in array AF.

Scope: **local**

Specified as: a fullword integer, where:

If (the process grid is  $p \times 1$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 502$ :

- For PDGTTRS,  $laf \geq 12P+3(MB\_A)$
- For PDDTTRS,  $laf \geq 12P+2(MB\_A)$ .

where, in the above formulas, P is the **actual** number of processes containing data.

If (the process grid is  $1 \times p$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 501$ , you would substitute NB\_A in place of MB\_A in the formulas above.

**Note:** In ScaLAPACK 1.5, PDDTTRS requires  $laf = 12P+3(NB\_A)$ . This value is greater than or equal to the value required by Parallel ESSL.

*work* has the following meaning:

If  $lwork = 0$ , *work* is ignored.

If  $lwork \neq 0$ , *work* is the work area used by this subroutine, where:

- If  $lwork \neq -1$ , the size of *work* is (at least) of length *lwork*.
- If  $lwork = -1$ , the size of *work* is (at least) of length 1.

Scope: **local**

Specified as: an area of storage containing numbers of data type indicated in Table 95 on page 548.

*lwork* is the number of elements in array WORK.

Scope:

- If  $lwork \geq 0$ , *lwork* is **local**
- If  $lwork = -1$ , *lwork* is **global**

Specified as: a fullword integer; where:

- If  $lwork = 0$ , PDGTTRS and PDDTTRS dynamically allocate the work area used by the subroutine. The work area is deallocated before control is returned to the calling program. This option is an extension to the ScaLAPACK standard.
- If  $lwork = -1$ , PDGTTRS and PDDTTRS perform a work area query and return the optimum size of *work* in *work*<sub>1</sub>. No computation is performed and the subroutine returns after error checking is complete.

- Otherwise, *lwork* must have the following value:

- For PDGTTRS,  $lwork \geq 12P+5(nrhs)$ .
- For PDDTTRS,  $lwork \geq 10P+4(nrhs)$

where, in the above formulas, *P* is the **actual** number of processes containing data.

*info* See On Return.

## On Return

*b* *b* is the updated local part of the global matrix *B*, containing the solution vectors.

Scope: **local**

Returned as: an LLD\_B by (at least) *nrhs* array, containing numbers of the data type indicated in Table 95 on page 548.

*work* is the work area used by this subroutine if *lwork*  $\neq$  0, where:

If *lwork*  $\neq$  0 and *lwork*  $\neq$  -1, the size of *work* is (at least) of length *lwork*.

If *lwork* = -1, the size of *work* is (at least) of length 1.

Scope: **local**

Returned as: an area of storage, containing numbers of data type indicated in Table 95 on page 548, where:

- If *lwork* = -1, the *work*<sub>1</sub> is set to the optimum *lwork* value needed.
- If *lwork*  $\geq$  1, the *work*<sub>1</sub> is set to the minimum *lwork* value needed.

Except for *work*<sub>1</sub>, the contents of *work* are overwritten on return.

*info* indicates that a successful computation or work area query occurred.

Scope: **global**

Returned as: a fullword integer; *info* = 0.

## Notes and Coding Rules

1. In your C program, argument *info* must be passed by reference.
2. The subroutine accepts lowercase letters for the *transa* argument.
3. The output from the factorization subroutines should be used only as input to the solve subroutines PDGTTRS and PDDTTRS, respectively.

The factored matrix *A* is stored in an internal format that depends on the number of processes.

The format of the output from PDDTTRF has changed. Therefore, the factorization and solve must be performed using Parallel ESSL Version 2 Release 1.2, or later.

The scalar data specified for input argument *n* must be the same for both PDGTTRF/PDDTTRF and PDGTTRS/PDDTTRS.

The global vectors for *dl*, *d*, *du*, *du2*, *ipiv*, and *af* input to PDGTTRS/PDDTTRS must be the same as the corresponding output arguments for PDGTTRF/PDDTTRF; and thus, the scalar data specified for *ia*, *desc\_a*, and *laf* must also be the same.

4. In all cases, follow these rules:
  - *ia* = *ib*
  - CTXT\_A = CTXT\_B
  - If (the process grid is  $p \times 1$  and DTYPE\_A = 1) or DTYPE\_A = 502, MB\_A = MB\_B.

- If (the process grid is  $1 \times p$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 501$ ,  $NB\_A = MB\_B$ .
- If  $DTYPE\_A=1$ , then:
  - For a  $p \times 1$  process grid (where  $p>1$ ),  $N\_A=1$ ,  $NB\_A \geq 1$ , and  $CSRC\_A=0$ .
  - For a  $1 \times p$  process grid (where  $p>1$ ),  $M\_A=1$ ,  $MB\_A \geq 1$ , and  $RSRC\_A=0$ .
  - For a  $1 \times 1$  process grid:
    - If  $N\_A=1$ ,  $NB\_A \geq 1$  and  $CSRC\_A=0$ .
    - If  $M\_A=1$ ,  $MB\_A \geq 1$  and  $RSRC\_A=0$ .
- If  $DTYPE\_B=1$ ,  $N\_B \geq nrhs$ ,  $NB\_B \geq 1$ , and  $CSRC\_B=0$ .
- Following are the consistent combinations of array descriptor types and process grids, where  $p$  is the number of processes in the process grid:

DTYPE_A	DTYPE_B	Process Grid
501	502	$p \times 1$ or $1 \times p$
502	502	$p \times 1$ or $1 \times p$
501	1	$p \times 1$
502	1	$p \times 1$
1	502	$p \times 1$ or $1 \times p$
1	1	$p \times 1$

- To determine the values of  $LOCp(n)$  used in the argument descriptions, see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28 for descriptor type-1 or “Determining the Number of Rows or Columns in Your Local Arrays” on page 32 for descriptor type-501 and type-502.
- $dl$ ,  $d$ ,  $du$ ,  $du2$ ,  $ipiv$ ,  $af$  and  $work$  must have no common elements; otherwise, results are unpredictable.
- The global general tridiagonal matrix  $A$  must be stored in tridiagonal storage mode and distributed over a one-dimensional process grid, using block-cyclic data distribution. See the section on block-cyclically distributing a tridiagonal matrix in “Matrices” on page 40.  
For more information on using block-cyclic data distribution, see “Specifying Block-Cyclically-Distributed Matrices for the Banded Linear Algebraic Equations” on page 29.
- Matrix  $B$  must be distributed over a one-dimensional process grid, using block-cyclic data distribution. For more information using block-cyclic data distribution, see “Specifying Block-Cyclically-Distributed Matrices for the Banded Linear Algebraic Equations” on page 29. Also, see the section on distributing the right-hand side matrix in “Matrices” on page 40.
- If  $lwork = -1$  on any process, it must equal  $-1$  on all processes. That is, if a subset of the processes specifies  $-1$  for the work area size, they must all specify  $-1$ .
- Although global matrices  $A$  and  $B$  may be block-cyclically distributed on a  $1 \times p$  or  $p \times 1$  process grid, the values of  $n$ ,  $ia$ ,  $ib$ ,  $MB\_A$  (if (the process grid is  $p \times 1$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 502$ ),  $NB\_A$  (if (the process grid is  $1 \times p$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 501$ ), must be chosen so that each process has at most one full or partial block of each of the global submatrices  $A$  and  $B$ .
- For global tridiagonal matrix  $A$ , use of the type-1 array descriptor is an extension to ScaLAPACK 1.5. If your application needs to run with both Parallel ESSL and ScaLAPACK 1.5, it is suggested that you use either a type-501 or a type-502 array descriptor for the matrix  $A$ .

## Error Conditions

### Computational Errors

None

**Note:** If the factorization performed by PDGTTRF or PDDTTRF failed because matrix  $A$  is singular or reducible, or is not diagonally dominant, respectively, the results returned by this subroutine are unpredictable. For details, see the *info* output argument for PDGTTRF or PDDTTRF.

### Resource Errors

Unable to allocate workspace

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. DTYPE\_A is invalid.
2. DTYPE\_B is invalid.

#### Stage 2:

1. CTEXT\_A is invalid.

#### Stage 3:

1. This subroutine was called from outside the process grid.

#### Stage 4:

**Note:** In the following error conditions:

- If  $M_A = 1$  and  $DTYPE_A = 1$ , a  $1 \times 1$  process grid is treated as a  $1 \times p$  process grid.
  - If  $N_A = 1$  and  $DTYPE_A = 1$ , a  $1 \times 1$  process grid is treated as a  $p \times 1$  process grid.
1. The process grid is not  $1 \times p$  or  $p \times 1$ .
  2.  $CTXT_A \neq CTXT_B$
  3. *transa*  $\neq$ 
    - 'N' for PDGTTRS
    - 'N', 'T', or 'C' for PDDTTRS
  4.  $n < 0$
  5.  $ia < 1$
  6.  $DTYPE_A = 1$  and  $M_A \neq 1$  and  $N_A \neq 1$   
 If (the process grid is  $1 \times p$  and  $DTYPE_A = 1$ ) or  $DTYPE_A = 501$ :
  7.  $N_A < 0$  and ( $n = 0$ );  $N_A < 1$  otherwise
  8.  $NB_A < 1$
  9.  $n > (NB_A)(p) - \text{mod}(ia-1, NB_A)$
  10.  $ia > N_A$  and ( $n > 0$ )
  11.  $ia+n-1 > N_A$  and ( $n > 0$ )
  12.  $CSRC_A < 0$  or  $CSRC_A \geq p$
  13.  $NB_A \neq MB_B$
  14.  $CSRC_A \neq RSRC_B$   
 If the process grid is  $1 \times p$  and  $DTYPE_A = 1$ :
  15.  $M_A \neq 1$
  16.  $MB_A < 1$
  17.  $RSRC_A \neq 0$   
 If (the process grid is  $p \times 1$  and  $DTYPE_A = 1$ ) or  $DTYPE_A = 502$ :
  18.  $M_A < 0$  and ( $n = 0$ );  $M_A < 1$  otherwise

19.  $MB\_A < 1$
20.  $n > (MB\_A)(p) - \text{mod}(ia-1, MB\_A)$
21.  $ia > MB\_A$  and  $(n > 0)$
22.  $ia+n-1 > M\_A$  and  $(n > 0)$
23.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
24.  $MB\_A \neq MB\_B$
25.  $RSRC\_A \neq RSRC\_B$   
 If the process grid is  $p \times 1$  and  $DTYPE\_A = 1$ :
  26.  $N\_A \neq 1$
  27.  $NB\_A < 1$
  28.  $CSRC\_A \neq 0$   
 In all cases:
    29.  $ia \neq ib$
    30.  $DTYPE\_B = 1$  and the process grid is  $1 \times p$  and  $p > 1$
    31.  $nrhs < 0$
    32.  $ib < 1$
    33.  $M\_B < 0$  and  $(n = 0)$ ;  $M\_B < 1$  otherwise
    34.  $MB\_B < 1$
    35.  $ib > M\_B$  and  $(n > 0)$
    36.  $ib+n-1 > M\_B$  and  $(n > 0)$
    37.  $RSRC\_B < 0$  or  $RSRC\_B \geq p$
    38.  $LLD\_B < \max(1, \text{LOCp}(M\_B))$   
 If  $DTYPE\_B = 1$ :
      39.  $N\_B < 0$  and  $(nrhs = 0)$ ;  $N\_B < 1$  otherwise
      40.  $N\_B < nrhs$
      41.  $NB\_B < 1$
      42.  $CSRC\_B \neq 0$   
 In all cases:
        43.  $laf < (\text{minimum value})$  (For the minimum value, see the *laf* argument description.)
        44.  $lwork \neq 0$ ,  $lwork \neq -1$ , and  $lwork < (\text{minimum value})$  (For the minimum value, see the *lwork* argument description.)

#### Stage 5:

Each of the following global input arguments are checked to determine whether its value is the same on all processes in the process grid:

1.  $n$  differs.
2.  $nrhs$  differs.
3.  $transa$  differs.
4.  $ia$  differs.
5.  $ib$  differs.
6.  $DTYPE\_A$  differs.  
 If  $DTYPE\_A = 1$  on all processes:
  7.  $M\_A$  differs.
  8.  $N\_A$  differs.
  9.  $MB\_A$  differs.
  10.  $NB\_A$  differs.
  11.  $RSRC\_A$  differs.
  12.  $CSRC\_A$  differs.  
 If  $DTYPE\_A = 501$  on all processes:
    13.  $N\_A$  differs.
    14.  $NB\_A$  differs.
    15.  $CSRC\_A$  differs.



If DTYPE\_A = 502 on all processes:

16. M\_A differs.
17. MB\_A differs.
18. RSRC\_A differs.

In all cases:

19. DTYPE\_B differs.

If DTYPE\_B = 1 on all processes:

20. M\_B differs.
21. N\_B differs.
22. MB\_B differs.
23. NB\_B differs.
24. RSRC\_B differs.
25. CSRC\_B differs.

If DTYPE\_B = 502 on all processes:

26. M\_B differs.
27. MB\_B differs.
28. RSRC\_B differs.

Also:

29. *lwork* = -1 on a subset of processes.

## Examples

### Example 1

This example shows how to solve the system  $AX=B$ , where matrix  $A$  is the same general tridiagonal matrix factored in “Example 1” on page 541 for PDGTTRF.

#### Notes:

1. The vectors *dl*, *d*, and *du*, output from PDGTTRF, are stored in an internal format that depends on the number of processes. These vectors are passed, unchanged, to the solve subroutine PDGTTRS.
2. The contents of these *du2* and *af* vectors, output from PDGTTRF, are not shown. These vectors are passed, unchanged, to the solve subroutine PDGTTRS.
3. Because *lwork* = 0, PDGTTRS dynamically allocates the work area used by this subroutine.

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 3
NPCOL = 1
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      TRANSA N  NRHS DL  D  DU  DU2  IA  DESC_A  IPIV  B  IB
      |      |  |   |   |   |   |   |      |   |   |
CALL PDGTTRS( N , 12 , 3 , DL , D , DU , DU2 , 1 , DESC_A , IPIV , B , 1 ,

      DESC_B  AF  LAF  WORK  LWORK INFO
      |      |   |   |   |   |   |
      DESC_B , AF , 48 , WORK , 0 , INFO )
```

	Desc_A
DTYPE_	502
CTXT_	<i>icontxt</i> <sup>1</sup>

## PDGTTRS and PDDTTRS

	Desc_A
M_	12
MB_	4
RSRC_	0
Not used	—
Reserved	—

### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.

	Desc_B
DTYPE_	502
CTXT_	<i>icontxt</i> <sup>1</sup>
M_	12
MB_	4
RSRC_	0
LLD_B	4
Reserved	—

### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.

Global vector *dl* with block size of 4:

$$\begin{array}{rcl}
 \text{B,D} & & 0 \\
 & & \left[ \begin{array}{c} . \\ 0.5 \\ 0.5 \\ 0.5 \\ \text{----} \\ 1.0 \\ 0.33 \\ 0.43 \\ 0.47 \\ \text{----} \\ 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \end{array} \right] \\
 0 & & \\
 & & \\
 1 & & \\
 & & \\
 2 & &
 \end{array}$$

Global vector *d* with block size of 4:

$$\begin{array}{rcl}
 \text{B,D} & & 0 \\
 & & \left[ \begin{array}{c} 0.5 \\ 0.5 \\ 0.5 \\ 2.0 \\ \text{----} \\ 0.33 \\ 0.43 \\ 0.47 \\ 2.07 \\ \text{----} \\ 2.07 \end{array} \right] \\
 0 & & \\
 & & \\
 1 & &
 \end{array}$$

$$2 \quad \left[ \begin{array}{c} 0.47 \\ 0.43 \\ 0.33 \end{array} \right]$$

Global vector *du* with block size of 4:

$$\begin{array}{cc} \text{B,D} & 0 \\ & \left[ \begin{array}{c} 2.0 \\ 2.0 \\ 2.0 \\ 2.0 \\ \text{----} \\ 2.0 \\ 2.0 \\ 2.0 \\ 2.0 \\ \text{----} \\ 0.93 \\ 0.86 \\ 0.67 \\ . \end{array} \right] \end{array}$$

Global vector *ipiv* with block size of 4:

$$\begin{array}{cc} \text{B,D} & 0 \\ & \left[ \begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ - \\ 0 \\ 0 \\ 0 \\ 0 \\ - \\ 0 \\ 0 \\ 0 \\ 0 \end{array} \right] \end{array}$$

The following is the 3 × 1 process grid:

$$\begin{array}{cc} \text{B,D} & 0 \\ \text{----} & \text{----} \\ 0 & P_{00} \\ \text{----} & \text{----} \\ 1 & P_{10} \\ \text{----} & \text{----} \\ 2 & P_{20} \end{array}$$

Local array DL with block size of 4:

$$\begin{array}{cc} \text{p,q} & 0 \\ \text{----} & \text{----} \\ & . \\ & 0.5 \\ 0 & 0.5 \\ & 0.5 \\ \text{----} & \text{----} \\ & 1.0 \\ & 0.33 \\ 1 & 0.43 \\ & 0.47 \end{array}$$

## PDGTTRS and PDDTTRS

	-----
	1.0
	1.0
2	1.0
	1.0

Local array D with block size of 4:

p,q	0
	-----
	0.5
	0.5
0	0.5
	2.0
	-----
	0.33
	0.43
1	0.47
	2.07
	-----
	2.07
	0.47
2	0.43
	0.33

Local array DU with block size of 4:

p,q	0
	-----
	2.0
	2.0
0	2.0
	2.0
	-----
	2.0
	2.0
1	2.0
	2.0
	-----
	0.93
	0.86
2	0.67
	.

Local array IPIV with block size of 4:

p,q	0
	-----
	0
	0
0	0
	0
	-----
	0
	0
1	0
	0
	-----
	0
	0
2	0
	0

Global matrix **B** with block size of 4:

B,D	0
	-----
	[ 46.0 6.0 4.0 ]

0	65.0	13.0	6.0
	59.0	19.0	6.0
	53.0	25.0	6.0
-----			
1	47.0	31.0	6.0
	41.0	37.0	6.0
	35.0	43.0	6.0
	29.0	49.0	6.0
-----			
2	23.0	55.0	6.0
	17.0	61.0	6.0
	11.0	67.0	6.0
	5.0	47.0	4.0

The following is the  $3 \times 1$  process grid:

B,D	0
-----	
0	P <sub>00</sub>
-----	
1	P <sub>10</sub>
-----	
2	P <sub>20</sub>

Local matrix  $B$  with block size of 4:

p,q	0
-----	
0	46.0 6.0 4.0
	65.0 13.0 6.0
	59.0 19.0 6.0
	53.0 25.0 6.0
-----	
1	47.0 31.0 6.0
	41.0 37.0 6.0
	35.0 43.0 6.0
	29.0 49.0 6.0
-----	
2	23.0 55.0 6.0
	17.0 61.0 6.0
	11.0 67.0 6.0
	5.0 47.0 4.0

**Output:**

Global matrix  $B$  with block size of 4:

B,D	0
-----	
0	12.0 1.0 1.0
	11.0 2.0 1.0
	10.0 3.0 1.0
	9.0 4.0 1.0
-----	
1	8.0 5.0 1.0
	7.0 6.0 1.0
	6.0 7.0 1.0
	5.0 8.0 1.0
-----	
2	4.0 9.0 1.0
	3.0 10.0 1.0
	2.0 11.0 1.0
	1.0 12.0 1.0

The following is the  $3 \times 1$  process grid:

## PDGTTRS and PDDTTRS

B,D	0
0	P <sub>00</sub>
1	P <sub>10</sub>
2	P <sub>20</sub>

Local matrix  $B$  with block size of 4:

p,q	0
0	12.0 1.0 1.0 11.0 2.0 1.0 10.0 3.0 1.0 9.0 4.0 1.0
1	8.0 5.0 1.0 7.0 6.0 1.0 6.0 7.0 1.0 5.0 8.0 1.0
2	4.0 9.0 1.0 3.0 10.0 1.0 2.0 11.0 1.0 1.0 12.0 1.0

The value of *info* is 0 on all processes.

### Example 2

This example shows how to solve the system  $AX=B$ , where matrix  $A$  is the same diagonally dominant general tridiagonal matrix factored in “Example 2” on page 547 for PDDTTRF. The input and/or output values for *dl*, *d*, *du*, *desc\_a*, and *info* in this example are the same as shown for “Example 1” on page 541.

#### Notes:

1. The vectors *dl*, *d*, and *du*, output from PDDTTRF, are stored in an internal format that depends on the number of processes. These vectors are passed, unchanged, to the solve subroutine PDDTTRS.
2. The contents of vector *af*, output from PDDTTRF, are not shown. This vector is passed, unchanged, to the solve subroutine PDDTTRS.
3. Because *lwork* = 0, PDDTTRS dynamically allocates the work area used by this subroutine.

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 3
NPCOL = 1
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      TRANSA N NRHS DL D DU IA DESC_A B IB DESC_B
      |      | | | | | | | | | |
CALL PDDTTRS( N , 12 , 3 , DL , D , DU , 1 , DESC_A , B , 1 , DESC_B ,

      AF LAF WORK LWORK INFO
      | | | | |
      AF , 44 , WORK , 0 , INFO )
```

## PDPTSV — Positive Definite Symmetric Tridiagonal Matrix Factorization and Solve

### Purpose

This subroutine solves the tridiagonal systems of linear equations,  $AX = B$ , where the positive definite symmetric tridiagonal matrix  $A$  is stored in parallel-symmetric-tridiagonal storage mode. In this description:

- $A$  represents the global positive definite symmetric tridiagonal submatrix  $A_{ia:ia+n-1, ia:ia+n-1}$ .
- $B$  represents the global general submatrix  $B_{ib:ib+n-1, 1:nrhs}$  containing the right-hand sides in its columns.
- $X$  represents the global general submatrix  $B_{ib:ib+n-1, 1:nrhs}$  containing the output solution vectors in its columns.

If  $n$  is 0, no computation is performed and the subroutine returns after doing some parameter checking. If  $n > 0$  and  $nrhs$  is 0, no solutions are computed and the subroutine returns after factoring the matrix.

See reference [54].

Table 100. Data Types

$d, e, B, work$	Subroutine
Long-precision real	PDPTSV

### Syntax

<b>Fortran</b>	CALL PDPTSV ( $n, nrhs, d, e, ia, desc\_a, b, ib, desc\_b, work, lwork, info$ )
<b>C and C++</b>	pdptsv ( $n, nrhs, d, e, ia, desc\_a, b, ib, desc\_b, work, lwork, info$ );

### On Entry

$n$  is the order of the positive definite symmetric tridiagonal matrix  $A$  and the number of rows in the general submatrix  $B$ , which contains the multiple right-hand sides.

Scope: **global**

Specified as: a fullword integer, where:

- If (the process grid is  $p \times 1$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 502$ ,  $0 \leq n \leq (MB\_A)(p) - \text{mod}(ia-1, MB\_A)$ .
- If (the process grid is  $1 \times p$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 501$ ,  $0 \leq n \leq (NB\_A)(p) - \text{mod}(ia-1, NB\_A)$ .

where  $p$  is the number of processes in a process grid.

$nrhs$  is the number of right-hand sides; that is, the number of columns in submatrix  $B$  used in the computation.

Scope: **global**

Specified as: a fullword integer;  $nrhs \geq 0$ .

$d$  is the local part of the global vector  $d$ . This identifies the **first element** of the local array D. This subroutine computes the location of the first element of the local subarray used, based on  $ia, desc\_a$ , and  $p$ ; therefore, the leading

$\text{LOCp}(ia+n-1)$  part of the local array  $D$  contains the local pieces of the leading  $ia+n-1$  part of the global vector.

The global vector  $d$  contains the main diagonal of the global positive definite symmetric tridiagonal submatrix  $A$  in elements  $ia$  through  $ia+n-1$ .

Scope: **local**

Specified as: a one-dimensional array of (at least) length  $\text{LOCp}(ia+n-1)$  containing numbers of the data type indicated in Table 100 on page 565. Details about block-cyclic data distribution of global matrix  $A$  are stored in  $desc\_a$ .

On output,  $D$  is overwritten; that is, the original input is not preserved.

$e$  is the local part of the global vector  $e$ . This identifies the **first element** of the local array  $E$ . This subroutine computes the location of the first element of the local subarray used, based on  $ia$ ,  $desc\_a$ , and  $p$ ; therefore, the leading  $\text{LOCp}(ia+n-1)$  part of the local array  $E$  contains the local pieces of the leading  $ia+n-1$  part of the global vector.

The global vector  $e$  contains the off-diagonal of the global positive definite symmetric tridiagonal submatrix  $A$  in elements  $ia$  through  $ia+n-2$ .

Scope: **local**

Specified as: a one-dimensional array of (at least) length  $\text{LOCp}(ia+n-1)$ , containing numbers of the data type indicated in Table 100 on page 565. Details about block-cyclic data distribution of global matrix  $A$  are stored in  $desc\_a$ .

On output,  $E$  is overwritten; that is, the original input is not preserved.

$ia$  is the row or column index of the global matrix  $A$ , identifying the first row or column of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer, where:

- If (the process grid is  $p \times 1$  and  $\text{DTYPE\_A} = 1$ ) or  $\text{DTYPE\_A} = 502$ ,  $1 \leq ia \leq M\_A$  and  $ia+n-1 \leq M\_A$ .
- If (the process grid is  $1 \times p$  and  $\text{DTYPE\_A} = 1$ ) or  $\text{DTYPE\_A} = 501$ ,  $1 \leq ia \leq N\_A$  and  $ia+n-1 \leq N\_A$ .

$desc\_a$  is the array descriptor for global matrix  $A$ . Because vectors are one-dimensional data structures, you may use a type-502, type-501, or type-1 array descriptor regardless of whether the process grid is  $p \times 1$  or  $1 \times p$ . For a type-502 array descriptor, the process grid is used as if it is a  $p \times 1$  process grid. For a type-501 array descriptor, the process grid is used as if it is a  $1 \times p$  process grid. For a type-1 array descriptor, the process grid is used as if it is either a  $p \times 1$  process grid or a  $1 \times p$  process grid. The following tables describe three types of array descriptors. For rules on using array descriptors, see “Notes and Coding Rules” on page 571.

Table 101. Type-502 Array Descriptor

$desc\_a$	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor Type	DTYPE_A=502 for $p \times 1$ or $1 \times p$ where $p$ is the number of processes in a process grid.	Global



Table 101. Type-502 Array Descriptor (continued)

<i>desc_a</i>	Name	Description	Limits	Scope
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	MB_A	Row block size	$MB\_A \geq 1$ and $0 \leq n \leq (MB\_A)(p) - \text{mod}(ia-1, MB\_A)$	Global
5	RSRC_A	The process row over which the first row of the global matrix is distributed	$0 \leq RSRC\_A < p$	Global
6	—	Not used by this subroutine.	—	—
7	—	Reserved	—	—

Specified as: an array of (at least) length 7, containing fullword integers.

Table 102. Type-1 Array Descriptor ( $p \times 1$  Process Grid)

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor Type	$DTYPE\_A = 1$ for $p \times 1$  where $p$ is the number of processes in a process grid.	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	$N\_A = 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$ and $0 \leq n \leq (MB\_A)(p) - \text{mod}(ia-1, MB\_A)$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global
7	RSRC_A	The process row over which the first row of the global matrix is distributed	$0 \leq RSRC\_A < p$	Global
8	CSRC_A	The process column over which the first column of the global matrix is distributed	$CSRC\_A = 0$	Global
9	—	Not used by this subroutine.	—	—

Specified as: an array of (at least) length 9, containing fullword integers.

Table 103. Type-501 Array Descriptor

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor Type	DTYPE_A=501 for $1 \times p$ or $p \times 1$ where $p$ is the number of processes in a process grid.	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	N_A	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
4	NB_A	Column block size	$NB\_A \geq 1$ and $0 \leq n \leq (NB\_A)(p) - \text{mod}(ia-1, NB\_A)$	Global
5	CSRC_A	The process column over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < p$	Global
6	—	Not used by this subroutine.	—	—
7	—	Reserved	—	—

Specified as: an array of (at least) length 7, containing fullword integers.

Table 104. Type-1 Array Descriptor ( $1 \times p$  Process Grid)

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A = 1 for $1 \times p$ where $p$ is the number of processes in a process grid.	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	$M\_A = 1$	Global
4	N_A	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$ and $0 \leq n \leq (NB\_A)(p) - \text{mod}(ia-1, NB\_A)$	Global
7	RSRC_A	The process row over which the first row of the global matrix is distributed	RSRC_A=0	Global
8	CSRC_A	The process column over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < p$	Global
9	—	Not used by this subroutine.	—	—

Specified as: an array of (at least) length 9, containing fullword integers.

$b$  is the local part of the global general matrix  $B$ , containing the multiple

right-hand sides of the system. This identifies the **first element** of the local array  $B$ . This subroutine computes the location of the first element of the local subarray used, based on  $ib$ ,  $desc\_b$ , and  $p$ ; therefore, the leading  $LOCp(ib+n-1)$  by  $nrhs$  part of the local array  $B$  must contain the local pieces of the leading  $ib+n-1$  by  $nrhs$  part of the global matrix.

Scope: **local**

Specified as: an  $LLD\_B$  by (at least)  $nrhs$  array, containing numbers of the data type indicated in Table 100 on page 565. Details about the block-cyclic data distribution of global matrix  $B$  are stored in  $desc\_b$ .

$ib$  is the row index of the global matrix  $B$ , identifying the first row of the submatrix  $B$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ib \leq M\_B$  and  $ib+n-1 \leq M\_B$ .

$desc\_b$  is the array descriptor for global matrix  $B$ , which may be type 502 or type 1, as described in the following tables. For type-502 array descriptor, the process grid is used as if it is a  $p \times 1$  process grid. For rules on using array descriptors, see “Notes and Coding Rules” on page 571.

$desc\_b$	Name	Description	Limits	Scope
1	DTYPE_B	Descriptor type	DTYPE_B = 502 for $p \times 1$ or $1 \times p$ where $p$ is the number of processes in a process grid.	Global
2	CTXT_B	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_B	Number of rows in the global matrix	If $n = 0$ : $M\_B \geq 0$ Otherwise: $M\_B \geq 1$	Global
4	MB_B	Row block size	$MB\_B \geq 1$ and $0 \leq n \leq (MB\_B)(p) - \text{mod}(ia-1, MB\_B)$	Global
5	RSRC_B	The process row over which the first row of the global matrix is distributed	$0 \leq RSRC\_B < p$	Global
6	LLD_B	Leading dimension	$LLD\_B \geq \max(1, LOCp(MB\_B))$	Local
7	—	Reserved	—	—

Specified as: an array of (at least) length 7, containing fullword integers.

$desc\_b$	Name	Description	Limits	Scope
1	DTYPE_B	Descriptor type	DTYPE_B = 1 for $p \times 1$ where $p$ is the number of processes in a process grid.	Global
2	CTXT_B	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global

<i>desc_b</i>	Name	Description	Limits	Scope
3	M_B	Number of rows in the global matrix	If $n = 0$ : M_B $\geq 0$ Otherwise: M_B $\geq 1$	Global
4	N_B	Number of columns in the global matrix	N_B $\geq nrhs$	Global
5	MB_B	Row block size	MB_B $\geq 1$ and $0 \leq n \leq (MB\_B)(p) - \text{mod}(ib-1, MB\_B)$	Global
6	NB_B	Column block size	NB_B $\geq 1$	Global
7	RSRC_B	The process row over which the first row of the global matrix is distributed	$0 \leq \text{RSRC\_B} < p$	Global
8	CSRC_B	The process column over which the first column of the global matrix is distributed	CSRC_B = 0	Global
9	LLD_B	Leading dimension	LLD_B $\geq \max(1, \text{LOCp}(MB\_B))$	Local

Specified as: an array of (at least) length 9, containing fullword integers.

*work* has the following meaning:

If  $lwork = 0$ , *work* is ignored.

If  $lwork \neq 0$ , *work* is the work area used by this subroutine, where:

- If  $lwork \neq -1$ , the size of *work* is (at least) of length *lwork*.
- If  $lwork = -1$ , the size of *work* is (at least) of length 1.

Scope: **local**

Specified as: an area of storage containing numbers of data type indicated in Table 100 on page 565.

*lwork* is the number of elements in array WORK.

Scope:

- If  $lwork \geq 0$ , *lwork* is **local**
- If  $lwork = -1$ , *lwork* is **global**

Specified as: a fullword integer; where:

- If  $lwork = 0$ , PDPTSV dynamically allocates the work area used by the subroutine. The work area is deallocated before control is returned to the calling program. This option is an extension to the ScaLAPACK standard.
- If  $lwork = -1$ , PDPTSV performs a work area query and return the optimum size of *work* in *work*<sub>1</sub>. No computation is performed and the subroutine returns after error checking is complete.
- Otherwise, if (the process grid is  $p \times 1$  and DTYPE\_A = 1) or DTYPE\_A = 502:
  - If  $nrhs \leq 1$ ,  $lwork \geq 10P + MB\_A + 10$ .
  - If  $nrhs > 1$ ,  $(lwork \geq (20 + 2\min(100, nrhs))P + 3(MB\_A) + 4(nrhs))$ .

where, in the above formulas, P is the **actual** number of processes containing data.

If (the process grid is  $1 \times p$  and DTYPE\_A = 1) or DTYPE\_A = 501, you would substitute NB\_A in place of MB\_A in the formulas above.

**Note:** In ScaLAPACK 1.5, PDPTSV requires  
 $lwork = 22P + 3MB\_A + 2\min(100, nrhs)P + 4(nrhs)$ . This value is  
greater than or equal to the value required by Parallel ESSL.

*info* See On Return.

### On Return

*d* is overwritten; that is, the original input is not preserved.

*e* is overwritten; that is, the original input is not preserved.

*b* *p* is the updated local part of the global matrix *B*, containing the solution vectors.

Scope: **local**

Returned as: an LLD\_B by (at least) *nrhs* array, containing numbers of the data type indicated in Table 100 on page 565.

*work* is the work area used by this subroutine if *lwork*  $\neq$  0, where:

If *lwork*  $\neq$  0 and *lwork*  $\neq$  -1, the size of *work* is (at least) of length *lwork*.

If *lwork* = -1, the size of *work* is (at least) of length 1.

Scope: **local**

Returned as: an area of storage, containing numbers of the data type indicated in Table 100 on page 565, where:

- If *lwork*  $\geq$  1, *work*<sub>1</sub> is set to the minimum *lwork* value needed.
- If *lwork* = -1, *work*<sub>1</sub> is set to the optimum *lwork* value needed.

Except for *work*<sub>1</sub>, the contents of *work* are overwritten on return.

*info* has the following meaning:

If *info* = 0, global submatrix *A* is positive definite, and the factorization completed successfully or the work area query completed successfully.

If  $1 \leq info \leq p$ , the portion of global submatrix *A* stored on process *info*-1 and factored locally, is not positive definite. A pivot element whose value is less than or equal to a small positive number was detected.

If *info* > *p*, the portion of global submatrix *A* stored on process *info*-*p*-1 representing interactions with other processes, is not positive definite. A pivot element whose value is less than or equal to a small positive number was detected.

If *info* > 0, the results of the computation are unpredictable.

Scope: **global**

Returned as: a fullword integer; *info*  $\geq$  0.

## Notes and Coding Rules

1. In your C program, argument *info* must be passed by reference.
2. If *n* > 0 and *nrhs* = 0, only the factorization is completed.
3. *d*, *e*, *B*, and *work* must have no common elements; otherwise, results are unpredictable.
4. In all cases, follow these rules:
  - *ia* = *ib*
  - CTXT\_A = CTXT\_B

- If (the process grid is  $p \times 1$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 502$ ,  $MB\_A = MB\_B$ .
- If (the process grid is  $1 \times p$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 501$ ,  $NB\_A = MB\_B$ .
- If  $DTYPE\_A=1$ , then:
  - For a  $p \times 1$  process grid (where  $p>1$ ),  $N\_A=1$ ,  $NB\_A \geq 1$ , and  $CSRC\_A=0$ .
  - For a  $1 \times p$  process grid (where  $p>1$ ),  $M\_A=1$ ,  $MB\_A \geq 1$ , and  $RSRC\_A=0$ .
  - For a  $1 \times 1$  process grid:
    - If  $N\_A=1$ ,  $NB\_A \geq 1$  and  $CSRC\_A=0$ .
    - If  $M\_A=1$ ,  $MB\_A \geq 1$  and  $RSRC\_A=0$ .
- If  $DTYPE\_B=1$ ,  $N\_B \geq nrhs$ ,  $NB\_B \geq 1$ , and  $CSRC\_B=0$ .
- Following are the consistent combinations of array descriptor types and process grids, where  $p$  is the number of processes in the process grid:

DTYPE_A	DTYPE_B	Process Grid
501	502	$p \times 1$ or $1 \times p$
502	502	$p \times 1$ or $1 \times p$
501	1	$p \times 1$
502	1	$p \times 1$
1	502	$p \times 1$ or $1 \times p$
1	1	$p \times 1$

- To determine the values of  $LOCp(n)$  used in the argument descriptions, see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28 for descriptor type-1 or “Determining the Number of Rows or Columns in Your Local Arrays” on page 32 for descriptor type-501 and type-502.
- The global symmetric tridiagonal matrix  $A$  must be positive definite. This subroutine uses the *info* argument to provide information about  $A$ , like ScaLAPACK. However, this subroutine also issues an error message, which differs from ScaLAPACK.
- The global positive definite symmetric tridiagonal matrix  $A$  must be stored in parallel-symmetric-tridiagonal storage mode and distributed over a one-dimensional process grid, using block-cyclic data distribution. See the section on block-cyclically distributing a tridiagonal matrix in “Matrices” on page 40.  
For more information on using block-cyclic data distribution, see “Specifying Block-Cyclically-Distributed Matrices for the Banded Linear Algebraic Equations” on page 29.
- Matrix  $B$  must be distributed over a one-dimensional process grid, using block-cyclic data distribution. For more information using block-cyclic data distribution, see “Specifying Block-Cyclically-Distributed Matrices for the Banded Linear Algebraic Equations” on page 29. Also, see the section on distributing the right-hand side matrix in “Matrices” on page 40.
- If  $lwork = -1$  on any process, it must equal  $-1$  on all processes. That is, if a subset of the processes specifies  $-1$  for the work area size, they must all specify  $-1$ .
- Although global matrices  $A$  and  $B$  may be block-cyclically distributed on a  $1 \times p$  or  $p \times 1$  process grid, the values of  $n$ ,  $ia$ ,  $ib$ ,  $MB\_A$  (if (the process grid is  $p \times 1$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 502$ ),  $NB\_A$  (if (the process grid is  $1 \times p$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 501$ ), must be chosen so that each process has at most one full or partial block of each of the global submatrices  $A$  and  $B$ .

11. For global tridiagonal matrix  $A$ , use of the type-1 array descriptor is an extension to ScaLAPACK 1.5. If your application needs to run with both Parallel ESSL and ScaLAPACK 1.5, it is suggested that you use either a type-501 or a type-502 array descriptor for the matrix  $A$ .

## Error Conditions

### Computational Errors

Matrix  $A$  is not positive definite. For details, see the description of the *info* argument.

### Resource Errors

Unable to allocate workspace

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. DTYPE\_A is invalid.
2. DTYPE\_B is invalid.

#### Stage 2:

1. CTEXT\_A is invalid.

#### Stage 3:

1. This subroutine was called from outside the process grid.

#### Stage 4:

**Note:** In the following error conditions:

- If  $M\_A = 1$  and  $DTYPE\_A = 1$ , a  $1 \times 1$  process grid is treated as a  $1 \times p$  process grid.
  - If  $N\_A = 1$  and  $DTYPE\_A = 1$ , a  $1 \times 1$  process grid is treated as a  $p \times 1$  process grid.
1. The process grid is not  $1 \times p$  or  $p \times 1$ .
  2.  $CTEXT\_A \neq CTEXT\_B$
  3.  $n < 0$
  4.  $ia < 1$
  5.  $DTYPE\_A = 1$  and  $M\_A \neq 1$  and  $N\_A \neq 1$   
If (the process grid is  $1 \times p$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 501$ :
  6.  $N\_A < 0$  and ( $n = 0$ );  $N\_A < 1$  otherwise
  7.  $NB\_A < 1$
  8.  $n > (NB\_A)(p) - \text{mod}(ia, NB\_A)$
  9.  $ia > N\_A$  and ( $n > 0$ )
  10.  $ia+n-1 > N\_A$  and ( $n > 0$ )
  11.  $CSRC\_A < 0$  or  $CSRC\_A \geq p$
  12.  $NB\_A \neq MB\_B$
  13.  $CSRC\_A \neq RSRC\_B$   
If the process grid is  $1 \times p$  and  $DTYPE\_A = 1$ :
  14.  $M\_A \neq 1$
  15.  $MB\_A < 1$
  16.  $RSRC\_A \neq 0$   
If (the process grid is  $p \times 1$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 502$ :
  17.  $M\_A < 0$  and ( $n = 0$ );  $M\_A < 1$  otherwise
  18.  $MB\_A < 1$
  19.  $n > (MB\_A)(p) - \text{mod}(ia, MB\_A)$

20.  $ia > M\_A$  and  $(n > 0)$
21.  $ia+n-1 > M\_A$  and  $(n > 0)$
22.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
23.  $MB\_A \neq MB\_B$
24.  $RSRC\_A \neq RSRC\_B$   
 If the process grid is  $p \times 1$  and  $DTYPE\_A = 1$ :
  25.  $N\_A \neq 1$
  26.  $NB\_A < 1$
  27.  $CSRC\_A \neq 0$   
 In all cases:
    28.  $ia \neq ib$
    29.  $DTYPE\_B = 1$  and the process grid is  $1 \times p$  and  $p > 1$
    30.  $nrhs < 0$
    31.  $ib < 1$
    32.  $M\_B < 0$  and  $(n = 0)$ ;  $M\_B < 1$  otherwise
    33.  $MB\_B < 1$
    34.  $ib > M\_B$  and  $(n > 0)$
    35.  $ib+n-1 > M\_B$  and  $(n > 0)$
    36.  $RSRC\_B \leq 0$  or  $RSRC\_B \geq p$
    37.  $LLD\_B < \max(1, LOCp(M\_B))$   
 If  $DTYPE\_B = 1$ :
      38.  $N\_B < 0$  and  $(nrhs = 0)$ ;  $N\_B < 1$  otherwise
      39.  $N\_B < nrhs$
      40.  $NB\_B < 1$
      41.  $CSRC\_B \neq 0$   
 In all cases:
        42.  $lwork \neq 0$ ,  $lwork \neq -1$ , and  $lwork < (\text{minimum value})$  (For the minimum value, see the *lwork* argument description.)

#### Stage 5:

Each of the following global input arguments are checked to determine whether its value is the same on all processes in the process grid:

1.  $n$  differs.
2.  $nrhs$  differs.
3.  $ia$  differs.
4.  $ib$  differs.
5.  $DTYPE\_A$  differs.  
 If  $DTYPE\_A = 1$  on all processes:
  6.  $M\_A$  differs.
  7.  $N\_A$  differs.
  8.  $MB\_A$  differs.
  9.  $NB\_A$  differs.
  10.  $RSRC\_A$  differs.
  11.  $CSRC\_A$  differs.  
 If  $DTYPE\_A = 501$  on all processes:
    12.  $N\_A$  differs.
    13.  $NB\_A$  differs.
    14.  $CSRC\_A$  differs.  
 If  $DTYPE\_A = 502$  on all processes:
      15.  $M\_A$  differs.
      16.  $MB\_A$  differs.
      17.  $RSRC\_A$  differs.
- In all cases:



18. DTYPE\_B differs.  
If DTYPE\_B = 1 on all processes:
19. M\_B differs.
20. N\_B differs.
21. MB\_B differs.
22. NB\_B differs.
23. RSRC\_B differs.
24. CSRC\_B differs.  
If DTYPE\_B = 502 on all processes:
25. M\_B differs.
26. MB\_B differs.
27. RSRC\_B differs.
- Also:
28. *lwork* = -1 on a subset of processes.

## Examples

### Example

This example shows a factorization of the positive definite symmetric tridiagonal matrix  $A$  of order 12:

$$\begin{bmatrix} 4.0 & 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 2.0 & 5.0 & 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 5.0 & 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 2.0 & 5.0 & 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 2.0 & 5.0 & 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 2.0 & 5.0 & 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 2.0 & 5.0 & 2.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 2.0 & 5.0 & 2.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 2.0 & 5.0 & 2.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 2.0 & 5.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 2.0 & 5.0 & 2.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 2.0 & 5.0 \end{bmatrix}$$

Matrix  $A$  is stored in parallel-symmetric-tridiagonal storage mode and is distributed over a  $1 \times 3$  process grid using block-cyclic distribution.

#### Notes:

1. On output, the vectors  $d$  and  $e$  are overwritten by this subroutine.
2. Notice **only one process grid was created**, even though, DTYPE\_A = 501 and DTYPE\_B = 502.
3. Because *lwork* = 0, this subroutine dynamically allocates the work area used by this subroutine.

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 1
NPCOL = 3
CALL BLACS_GET (0, 0, ICONXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      N  NRHS  D   E   IA   DESC_A  B  IB   DESC_B  WORK  LWORK  INFO
      |   |   |   |   |   |       |   |   |       |   |   |
CALL PDPTSV( 12 , 3 , D , E , 1 , DESC_A , B , 1 , DESC_B , WORK , 0 , INFO)
```

	Desc_A
DTYPE_	501
CTXT_	<i>icontxt</i> <sup>1</sup>
N_	12
NB_	4
CSRC_	0
Not used	—
Reserved	—

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.

	Desc_B
DTYPE_	502
CTXT_	<i>icontxt</i> <sup>1</sup>
M_	12
MB_	4
RSRC_	0
LLD_B	4
Reserved	—

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.

Global vector *d* with block size of 4:

B,D                      0                                      1                                      2

0     $\left[ \begin{array}{cccc|cccc|cccc} 4.0 & 5.0 & 5.0 & 5.0 & 5.0 & 5.0 & 5.0 & 5.0 & 5.0 & 5.0 & 5.0 & 5.0 \end{array} \right]$

Global vector *e* with block size of 4:

B,D                      0                                      1                                      2

0     $\left[ \begin{array}{cccc|cccc|cccc} 2.0 & 2.0 & 2.0 & 2.0 & 2.0 & 2.0 & 2.0 & 2.0 & 2.0 & 2.0 & 2.0 & . \end{array} \right]$

The following is the  $1 \times 3$  process grid:

B,D	0	1	2
-----	-----	-----	-----
0	P <sub>00</sub>	P <sub>01</sub>	P <sub>02</sub>

Local array D with block size of 4:

p,q	0	1	2
-----	-----	-----	-----
0	4.0 5.0 5.0 5.0	5.0 5.0 5.0 5.0	5.0 5.0 5.0 5.0

Local array E with block size of 4:

p,q	0				1				2			
0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	.

Global matrix  $B$  with a block size of 4:

p,q	0		
0	70.0	8.0	6.0
	99.0	18.0	9.0
	90.0	27.0	9.0
	81.0	36.0	9.0
1	72.0	45.0	9.0
	63.0	54.0	9.0
	54.0	63.0	9.0
	45.0	72.0	9.0
2	36.0	81.0	9.0
	27.0	90.0	9.0
	18.0	99.0	9.0
	9.0	82.0	7.0

The following is the  $1 \times 3$  process grid:

B,D	0	1	2
0	$P_{00}$	$P_{01}$	$P_{02}$

Local matrix  $B$  with a block size of 4:

p,q	0			1			2		
0	70.0	8.0	6.0	72.0	45.0	9.0	36.0	81.0	9.0
	99.0	18.0	9.0	63.0	54.0	9.0	27.0	90.0	9.0
	90.0	27.0	9.0	54.0	63.0	9.0	18.0	99.0	9.0
	81.0	36.0	9.0	45.0	72.0	9.0	9.0	82.0	7.0

**Output:**

Global matrix  $B$  with a block size of 4:

p,q	0		
0	12.0	1.0	1.0
	11.0	2.0	1.0
	10.0	3.0	1.0
	9.0	4.0	1.0
1	8.0	5.0	1.0
	7.0	6.0	1.0
	6.0	7.0	1.0
	5.0	8.0	1.0
2	4.0	9.0	1.0
	3.0	10.0	1.0
	2.0	11.0	1.0
	1.0	12.0	1.0

The following is the  $1 \times 3$  process grid:

B,D	0	1	2
0	$P_{00}$	$P_{01}$	$P_{02}$

Local matrix  $B$  with a block size of 4:

## PDPTSV

p,q	0			1			2		
0	12.0	1.0	1.0	8.0	5.0	1.0	4.0	9.0	1.0
	11.0	2.0	1.0	7.0	6.0	1.0	3.0	10.0	1.0
	10.0	3.0	1.0	6.0	7.0	1.0	2.0	11.0	1.0
	9.0	4.0	1.0	5.0	8.0	1.0	1.0	12.0	1.0

The value of *info* is 0 on all processes.

## PDPTTRF — Positive Definite Symmetric Tridiagonal Matrix Factorization

### Purpose

This subroutine factors the positive definite symmetric tridiagonal matrix  $A$ , stored in parallel-symmetric-tridiagonal storage mode, where, in this description,  $A$  represents the global positive definite symmetric tridiagonal submatrix

$$A_{ia:ia+n-1, ia:ia+n-1}$$

To solve a tridiagonal system of linear equations with multiple right-hand sides, follow the call to PDPTTRF with one or more calls to PDPTTRS. The output from this factorization subroutine should be used only as input to the solve subroutine PDPTTRS.

If  $n = 0$ , no computation is performed and the subroutine returns after doing some parameter checking. See reference [54].

Table 105. Data Types

$d, e, af, work$	Subroutine
Long-precision real	PDPTTRF

### Syntax

Fortran	CALL PDPTTRF ( $n, d, e, ia, desc\_a, af, laf, work, lwork, info$ )
C and C++	pdpttrf ( $n, d, e, ia, desc\_a, af, laf, work, lwork, info$ );

### On Entry

$n$  is the order of the positive definite symmetric tridiagonal matrix  $A$ .

Scope: **global**

Specified as: a fullword integer, where:

- If (the process grid is  $p \times 1$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 502$ ,  $0 \leq n \leq (MB\_A)(p) - \text{mod}(ia-1, MB\_A)$ .
- If (the process grid is  $1 \times p$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 501$ ,  $0 \leq n \leq (NB\_A)(p) - \text{mod}(ia-1, NB\_A)$ .

where  $p$  is the number of processes in a process grid.

$d$  is the local part of the global vector  $d$ . This identifies the **first element** of the local array  $D$ . This subroutine computes the location of the first element of the local subarray used, based on  $ia, desc\_a$ , and  $p$ ; therefore, the leading  $LOCp(ia+n-1)$  part of the local array  $D$  contains the local pieces of the leading  $ia+n-1$  part of the global vector.

The global vector  $d$  contains the main diagonal of the global positive definite symmetric tridiagonal submatrix  $A$  in elements  $ia$  through  $ia+n-1$ .

Scope: **local**

Specified as: a one-dimensional array of (at least) length  $LOCp(ia+n-1)$ , containing numbers of the data type indicated in Table 105. Details about block-cyclic data distribution of global matrix  $A$  are stored in  $desc\_a$ .

On output, D is overwritten; that is, the original input is not preserved.

*e* is the local part of the global vector *e*. This identifies the **first element** of the local array E. This subroutine computes the location of the first element of the local subarray used, based on *ia*, *desc\_a*, and *p*; therefore, the leading LOCp(*ia+n-1*) part of the local array E contains the local pieces of the leading *ia+n-1* part of the global vector.

The global vector *e* contains the off-diagonal of the global positive definite symmetric tridiagonal submatrix *A* in elements *ia* through *ia+n-2*.

Scope: **local**

Specified as: a one-dimensional array of (at least) length LOCp(*ia+n-1*), containing numbers of the data type indicated in Table 105 on page 579. Details about block-cyclic data distribution of global matrix *A* are stored in *desc\_a*.

On output, E is overwritten; that is, the original input is not preserved.

*ia* is the row or column index of the global matrix *A*, identifying the first row or column of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer; where:

- If (the process grid is  $p \times 1$  and DTYPE\_A = 1) or DTYPE\_A = 502,  $1 \leq ia \leq M\_A$  and  $ia+n-1 \leq M\_A$ .
- If (the process grid is  $1 \times p$  and DTYPE\_A = 1) or DTYPE\_A = 501,  $1 \leq ia \leq N\_A$  and  $ia+n-1 \leq N\_A$ .

*desc\_a* is the array descriptor for global matrix *A*. Because vectors are one-dimensional data structures, you may use a type-502, type-501, or type-1 array descriptor regardless of whether the process grid is  $p \times 1$  or  $1 \times p$ . For a type-502 array descriptor, the process grid is used as if it is a  $p \times 1$  process grid. For a type-501 array descriptor, the process grid is used as if it is a  $1 \times p$  process grid. For a type-1 array descriptor, the process grid is used as if it is either a  $p \times 1$  process grid or a  $1 \times p$  process grid. The following tables describe three types of array descriptors.

Table 106. Type-502 Array Descriptor

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor Type	DTYPE_A=502 for $p \times 1$ or $1 \times p$  where <i>p</i> is the number of processes in a process grid.	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	MB_A	Row block size	$MB\_A \geq 1$ and $0 \leq n \leq (MB\_A)(p) - \text{mod}(ia-1, MB\_A)$	Global
5	RSRC_A	The process row over which the first row of the global matrix is distributed	$0 \leq \text{RSRC\_A} < p$	Global
6	—	Not used by this subroutine.	—	—

Table 106. Type-502 Array Descriptor (continued)

<i>desc_a</i>	Name	Description	Limits	Scope
7	—	Reserved	—	—

Specified as: an array of (at least) length 7, containing fullword integers.

Table 107. Type-1 Array Descriptor ( $p \times 1$  Process Grid)

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor Type	DTYPE_A = 1 for $p \times 1$  where $p$ is the number of processes in a process grid.	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	$N\_A = 1$	
5	MB_A	Row block size	$MB\_A \geq 1$ and $0 \leq n \leq (MB\_A)(p) - \text{mod}(ia-1, MB\_A)$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global
7	RSRC_A	The process row over which the first row of the global matrix is distributed	$0 \leq \text{RSRC\_A} < p$	Global
8	CSRC_A	The process column over which the first column of the global matrix is distributed	$\text{CSRC\_A} = 0$	Global
9	—	Not used by this subroutine.	—	—

Specified as: an array of (at least) length 9, containing fullword integers.

Table 108. Type-501 Array Descriptor

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor Type	DTYPE_A=501 for $1 \times p$ or $p \times 1$  where $p$ is the number of processes in a process grid.	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	N_A	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
4	NB_A	Column block size	$NB\_A \geq 1$ and $0 \leq n \leq (NB\_A)(p) - \text{mod}(ia-1, NB\_A)$	Global

Table 108. Type-501 Array Descriptor (continued)

<i>desc_a</i>	Name	Description	Limits	Scope
5	CSRC_A	The process column over which the first column of the global matrix is distributed	$0 \leq \text{CSRC\_A} < p$	Global
6	—	Not used by this subroutine.	—	—
7	—	Reserved	—	—

Specified as: an array of (at least) length 7, containing fullword integers.

Table 109. Type-1 Array Descriptor ( $1 \times p$  Process Grid)

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A = 1 for $1 \times p$ where $p$ is the number of processes in a process grid.	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	$M\_A = 1$	Global
4	N_A	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$ and $0 \leq n \leq (NB\_A)(p) - \text{mod}(ia-1, NB\_A)$	Global
7	RSRC_A	The process row over which the first row of the global matrix is distributed	$RSRC\_A = 0$	Global
8	CSRC_A	The process column over which the first column of the global matrix is distributed	$0 \leq \text{CSRC\_A} < p$	Global
9	—	Not used by this subroutine.	—	—

Specified as: an array of (at least) length 9, containing fullword integers.

*af* See On Return.

*laf* is the number of elements in array AF.

Scope: **local**

Specified as: a fullword integer, where:

- If (the process grid is  $p \times 1$  and DTYPE\_A = 1) or DTYPE\_A = 502,  
 $laf \geq 12P+3(MB\_A)$ .
- If (the process grid is  $1 \times p$  and DTYPE\_A = 1) or DTYPE\_A = 501,  
 $laf \geq 12P+3(NB\_A)$ .

where, in the formulas above, P is the **actual** number of processes containing data.

*work* has the following meaning:



If  $lwork = 0$ ,  $work$  is ignored.

If  $lwork \neq 0$ ,  $work$  is the work area used by this subroutine, where:

- If  $lwork \neq -1$ , the size of  $work$  is (at least) of length  $lwork$ .
- If  $lwork = -1$ , the size of  $work$  is (at least) of length 1.

Scope: **local**

Specified as: an area of storage containing numbers of data type indicated in Table 105 on page 579.

$lwork$  is the number of elements in array  $WORK$ .

Scope:

- If  $lwork \geq 0$ ,  $lwork$  is **local**
- If  $lwork = -1$ ,  $lwork$  is **global**

Specified as: a fullword integer; where:

- If  $lwork = 0$ , PDPTTRF dynamically allocates the work area used by the subroutine. The work area is deallocated before control is returned to the calling program. This option is an extension to the ScaLAPACK standard.
- If  $lwork = -1$ , PDPTTRF performs a work area query and return the optimum size of  $work$  in  $work_1$ . No computation is performed and the subroutine returns after error checking is complete.
- Otherwise,  $lwork \geq 8P$ , where  $P$  is the **actual** number of processes containing data.

*info* See On Return.

## On Return

$d$   $d$  is the updated local part of the global vector  $d$ , containing part of the factorization.

Scope: **local**

Returned as: a one-dimensional array of (at least) length  $LOCp(ia+n-1)$ , containing numbers of the data type indicated in Table 105 on page 579.

On output,  $D$  is overwritten; that is, the original input is not preserved.

$e$   $e$  is the updated local part of the global vector  $e$ , containing part of the factorization.

Scope: **local**

Returned as: a one-dimensional array of (at least) length  $LOCp(ia+n-1)$ , containing numbers of the data type indicated in Table 105 on page 579.

On output,  $E$  is overwritten; that is, the original input is not preserved.

$af$  is a work area used by this subroutine and contains part of the factorization. Its size is specified by  $laf$ .

Scope: **local**

Returned as: a one-dimensional array of (at least) length  $laf$ , containing numbers of the data type indicated in Table 105 on page 579.

$work$  is the work area used by this subroutine if  $lwork \neq 0$ , where:

- If  $lwork \neq 0$  and  $lwork \neq -1$ , the size of  $work$  is (at least) of length  $lwork$ .
- If  $lwork = -1$ , the size of  $work$  is (at least) of length 1.

Scope: **local**

Returned as: an area of storage, containing numbers of data type indicated in Table 105 on page 579, where:

- If  $lwork \geq 1$ , the  $work_1$  is set to the minimum  $lwork$  value needed.
- If  $lwork = -1$ , the  $work_1$  is set to the optimum  $lwork$  value needed.

Except for  $work_1$ , the contents of  $work$  are overwritten on return.

*info* has the following meaning:

If  $info = 0$ , global submatrix  $A$  is positive definite, and the factorization completed successfully or the work area query completed successfully.

If  $1 \leq info \leq p$ , the portion of global submatrix  $A$  stored on process  $info-1$  and factored locally, is not positive definite. A pivot element whose value is less than or equal to a small positive number was detected.

If  $info > p$ , the portion of global submatrix  $A$  stored on process  $info-p-1$  representing interactions with other processes, is not positive definite. A pivot element whose value is less than or equal to a small positive number was detected.

If  $info > 0$ , the factorization is completed; however, if you call PDPTTRS with these factors, the results of the computation are unpredictable.

Scope: **global**

Returned as: a fullword integer;  $info \geq 0$ .

## Notes and Coding Rules

1. In your C program, argument *info* must be passed by reference.
2. The output from these factorization subroutines should be used only as input to the solve subroutine PDPTTRS.

The factored matrix  $A$  is stored in an internal format that depends on the number of processes.

The scalar data specified for input argument  $n$  must be the same for both PDPTTRF and PDPTTRS.

The global vectors for  $d$ ,  $e$ , and  $af$  input to PDPTTRS must be the same as the corresponding output arguments for PDPTTRF; and thus, the scalar data specified for  $ia$ ,  $desc_a$ , and  $laf$  must also be the same.

3. In all cases, follow these rules:
  - If  $DTYPE\_A=1$ , then:
    - For a  $p \times 1$  process grid (where  $p>1$ ),  $N\_A=1$ ,  $NB\_A \geq 1$ , and  $CSRC\_A=0$ .
    - For a  $1 \times p$  process grid (where  $p>1$ ),  $M\_A=1$ ,  $MB\_A \geq 1$ , and  $RSRC\_A=0$ .
    - For a  $1 \times 1$  process grid:
      - If  $N\_A=1$ ,  $NB\_A \geq 1$  and  $CSRC\_A=0$ .
      - If  $M\_A=1$ ,  $MB\_A \geq 1$  and  $RSRC\_A=0$ .
  - Following are the consistent combinations of array descriptor types and process grids, where  $p$  is the number of processes in the process grid:

<b>DTYPE_A</b>	<b>Process Grid</b>
501	$p \times 1$ or $1 \times p$
502	$p \times 1$ or $1 \times p$
1	$p \times 1$ or $1 \times p$

4. To determine the values of  $LOCp(n)$  used in the argument descriptions, see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28

page 28 for descriptor type-1 or “Determining the Number of Rows or Columns in Your Local Arrays” on page 32 for descriptor type-501 and type-502.

5.  $d$ ,  $e$ ,  $af$ , and  $work$  must have no common elements; otherwise, results are unpredictable.
6. The global symmetric tridiagonal matrix  $A$  must be positive definite. This subroutine uses the *info* argument to provide information about  $A$ , like ScaLAPACK. However, this subroutine also issues an error message, which differs from ScaLAPACK.
7. The global positive definite symmetric tridiagonal matrix  $A$  must be stored in parallel-symmetric-tridiagonal storage mode and distributed over a one-dimensional process grid, using block-cyclic data distribution. See the section on block-cyclically distributing a tridiagonal matrix in “Matrices” on page 40.  
For more information on using block-cyclic data distribution, see “Specifying Block-Cyclically-Distributed Matrices for the Banded Linear Algebraic Equations” on page 29.
8. If  $lwork = -1$  on any process, it must equal  $-1$  on all processes. That is, if a subset of the processes specifies  $-1$  for the work area size, they must all specify  $-1$ .
9. Although global matrix  $A$  may be block-cyclically distributed on a  $1 \times p$  or  $p \times 1$  process grid, the values of  $n$ ,  $ia$ ,  $MB\_A$  (if (the process grid is  $p \times 1$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 502$ ),  $NB\_A$  (if (the process grid is  $1 \times p$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 501$ ), must be chosen so that each process has at most one full or partial block of global submatrix  $A$ .
10. For global tridiagonal matrix  $A$ , use of the type-1 array descriptor is an extension to ScaLAPACK 1.5. If your application needs to run with both Parallel ESSL and ScaLAPACK 1.5, it is suggested that you use either a type-501 or a type-502 array descriptor for the matrix  $A$ .

## Error Conditions

### Computational Errors

Matrix  $A$  is not positive definite. For details, see the description of the *info* argument.

### Resource Errors

Unable to allocate workspace

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1.  $DTYPE\_A$  is invalid.

#### Stage 2:

1.  $CTXT\_A$  is invalid.

#### Stage 3:

1. This subroutine was called from outside the process grid.

#### Stage 4:

**Note:** In the following error conditions:

- If  $M\_A = 1$  and  $DTYPE\_A = 1$ , a  $1 \times 1$  process grid is treated as a  $1 \times p$  process grid.
  - If  $N\_A = 1$  and  $DTYPE\_A = 1$ , a  $1 \times 1$  process grid is treated as a  $p \times 1$  process grid.
1. The process grid is not  $1 \times p$  or  $p \times 1$ .
  2.  $n < 0$
  3.  $ia < 1$
  4.  $DTYPE\_A = 1$  and  $M\_A \neq 1$  and  $N\_A \neq 1$   
 If (the process grid is  $1 \times p$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 501$ :
    5.  $N\_A < 0$  and ( $n = 0$ );  $N\_A < 1$  otherwise
    6.  $NB\_A < 1$
    7.  $n > (NB\_A)(p) - \text{mod}(ia-1, NB\_A)$
    8.  $ia > N\_A$  and ( $n > 0$ )
    9.  $ia+n-1 > N\_A$  and ( $n > 0$ )
  10.  $CSRC\_A < 0$  or  $CSRC\_A \geq p$   
 If the process grid is  $1 \times p$  and  $DTYPE\_A = 1$ :
    11.  $M\_A \neq 1$
    12.  $MB\_A < 1$
    13.  $RSRC\_A \neq 0$   
 If (the process grid is  $p \times 1$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 502$ :
      14.  $M\_A < 0$  and ( $n = 0$ );  $M\_A < 1$  otherwise
      15.  $MB\_A < 1$
      16.  $n > (MB\_A)(p) - \text{mod}(ia-1, MB\_A)$
      17.  $ia > M\_A$  and ( $n > 0$ )
      18.  $ia+n-1 > M\_A$  and ( $n > 0$ )
    19.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$   
 If the process grid is  $p \times 1$  and  $DTYPE\_A = 1$ :
      20.  $N\_A \neq 1$
      21.  $NB\_A < 1$
      22.  $CSRC\_A \neq 0$
  - In all cases:
    23.  $laf < (\text{minimum value})$  (For the minimum value, see the *laf* argument description.)
    24.  $lwork \neq 0$ ,  $lwork \neq -1$ , and  $lwork < (\text{minimum value})$  (For the minimum value, see the *lwork* argument description.)

#### Stage 5:

Each of the following global input arguments are checked to determine whether its value is the same on all processes in the process grid:

1.  $n$  differs.
2.  $ia$  differs.
3.  $DTYPE\_A$  differs.  
 If  $DTYPE\_A = 1$  on all processes:
  4.  $M\_A$  differs.
  5.  $N\_A$  differs.
  6.  $MB\_A$  differs.
  7.  $NB\_A$  differs.
  8.  $RSRC\_A$  differs.
  9.  $CSRC\_A$  differs.
- If  $DTYPE\_A = 501$  on all processes:
  10.  $N\_A$  differs.
  11.  $NB\_A$  differs.
  12.  $CSRC\_A$  differs.

If DTYPE\_A = 502 on all processes:

13. M\_A differs.
14. MB\_A differs.
15. RSRC\_A differs.

Also:

16. *lwork* = -1 on a subset of processes.

## Examples

### Example

This example shows a factorization of the positive definite symmetric tridiagonal matrix *A* of order 12.

$$\begin{bmatrix} 4.0 & 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 2.0 & 5.0 & 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 5.0 & 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 2.0 & 5.0 & 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 2.0 & 5.0 & 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 2.0 & 5.0 & 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 2.0 & 5.0 & 2.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 2.0 & 5.0 & 2.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 2.0 & 5.0 & 2.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 2.0 & 5.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 2.0 & 5.0 & 2.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 2.0 & 5.0 \end{bmatrix}$$

Matrix *A* is stored in parallel-symmetric-tridiagonal storage mode and is distributed over a  $3 \times 1$  process grid using block-cyclic distribution.

#### Notes:

1. The vectors *d* and *e*, output from PDPTRF, are stored in an internal format that depends on the number of processes. These vectors are passed, unchanged, to the solve subroutine PDPTTRS.
2. The contents of the *af* vector, output from PDPTRF, is not shown. This vector is passed, unchanged, to the solve subroutine PDPTTRS.
3. Because *lwork* = 0, this subroutine dynamically allocates the work area used by this subroutine.

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 3
NPCOL = 1
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      N   D   E   IA   DESC_A   AF   LAF   WORK   LWORK   INFO
      |   |   |   |   |       |   |   |   |   |   |
CALL PDPTRF( 12 , D , E , 1 , DESC_A , AF , 48 , WORK , 0 , INFO )
```

	Desc_A
DTYPE_	502
CTXT_	<i>icontxt</i> <sup>1</sup>
M_	12
MB_	4
RSRC_	0

	Desc_A
Not used	—
Reserved	—

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.

Global vector *d* with block size of 4:

B,D	0
	4.0
	5.0
0	5.0
	5.0
	---
	5.0
	5.0
1	5.0
	5.0
	---
	5.0
	5.0
2	5.0
	5.0

Global vector *e* with block size of 4:

B,D	0
	2.0
	2.0
0	2.0
	2.0
	---
	2.0
	2.0
1	2.0
	2.0
	---
	2.0
	2.0
2	2.0
	.

The following is the  $3 \times 1$  process grid:

B,D	0
-----	-----
0	P <sub>00</sub>
-----	-----
1	P <sub>10</sub>
-----	-----
2	P <sub>20</sub>

Local array D with block size of 4:

p,q	0
-----	-----
	4.0
	5.0
0	5.0
	5.0

-----	
1	5.0
	5.0
	5.0
	5.0
-----	
2	5.0
	5.0
	5.0
	5.0

Local array E with block size of 4:

p,q	0
-----	
0	2.0
	2.0
	2.0
	2.0
-----	
1	2.0
	2.0
	2.0
	2.0
-----	
2	2.0
	2.0
	2.0
	.

**Output:**

Global vector  $d$  with block size of 4:

B,D	0
0	.25
	.25
	.25
	4.0
-----	
1	.2
	.24
	.25
	4.01
-----	
2	4.01
	.25
	.24
	.2

Global vector  $e$  with block size of 4:

B,D	0
0	2.0
	2.0
	2.0
	2.0
-----	
1	2.0
	2.0
	2.0
	2.0
-----	
	.49

$$2 \quad \begin{bmatrix} .48 \\ .4 \\ . \end{bmatrix}$$

The following is the  $3 \times 1$  process grid:

B,D	0
0	P <sub>00</sub>
1	P <sub>10</sub>
2	P <sub>20</sub>

Local array D with block size of 4:

p,q	0
0	.25 .25 .25 4.0
1	.2 .24 .25 4.01
2	4.01 .25 .24 .2

Local array E with block size of 4:

p,q	0
0	2.0 2.0 2.0 2.0
1	2.0 2.0 2.0 2.0
2	.49 .48 .4 .

The value of *info* is 0 on all processes.



## PDPTTRS — Positive Definite Symmetric Tridiagonal Matrix Solve

### Purpose

This subroutine solves the following tridiagonal systems of linear equations for multiple right-hand sides, using the positive definite symmetric tridiagonal matrix  $A$ , where  $A$  is stored in parallel-symmetric-tridiagonal storage mode:

- $AX = B$

In this subroutine:

- $A$  represents the global positive definite symmetric tridiagonal submatrix  $A_{ia:ia+n-1, ia:ia+n-1}$ .
- $B$  represents the global general submatrix  $B_{ib:ib+n-1, 1:nrhs}$  containing the right-hand sides in its columns.
- $X$  represents the global general submatrix  $B_{ib:ib+n-1, 1:nrhs}$  containing the output solution vectors in its columns.

This subroutine uses the results of the factorization of matrix  $A$ , produced by a preceding call to PDPTTRF. The output from PDPTTRF should be used only as input to this solve subroutine.

If  $n = 0$  or  $nrhs = 0$ , no computation is performed and the subroutine returns after doing some parameter checking. See reference [54].

Table 110. Data Types

$d, e, B, af, work$	Subroutine
Long-precision real	PDPTTRS

### Syntax

<b>Fortran</b>	CALL PDPTTRS ( $n, nrhs, d, e, ia, desc\_a, b, ib, desc\_b, af, laf, work, lwork, info$ )
<b>C and C++</b>	pdpttrs ( $n, nrhs, d, e, ia, desc\_a, b, ib, desc\_b, af, laf, work, lwork, info$ );

### On Entry

$n$  is the order of the positive definite symmetric tridiagonal submatrix  $A$  and the number of rows in the general submatrix  $B$ , which contains the multiple right-hand sides.

Scope: **global**

Specified as: a fullword integer, where:

- If (the process grid is  $p \times 1$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 502$ ,  $0 \leq n \leq (MB\_A)(p) - \text{mod}(ia-1, MB\_A)$ .
- If (the process grid is  $1 \times p$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 501$ ,  $0 \leq n \leq (NB\_A)(p) - \text{mod}(ia-1, NB\_A)$ .

where  $p$  is the number of processes in a process grid.

$nrhs$  is the number of right-hand sides; that is, the number of columns in submatrix  $B$  used in the computation.

Scope: **global**

Specified as: a fullword integer;  $nrhs \geq 0$ .

*d* is the local part of the global vector *d*, containing part of the factorization produced from a preceding call to PDPTTRF. This identifies the **first element** of the local array D. This subroutine computes the location of the first element of the local subarray used, based on *ia*, *desc\_a*, and *p*; therefore, the leading LOCp(*ia+n-1*) part of the local array D contains the local pieces of the leading *ia+n-1* part of the global vector.

Scope: **local**

Specified as: a one-dimensional array of (at least) length LOCp(*ia+n-1*), containing numbers of the data type indicated in Table 110 on page 591. Details about block-cyclic data distribution of global matrix *A* are stored in *desc\_a*.

*e* is the local part of the global vector *e*, containing part of the factorization produced from a preceding call to PDPTTRF. This identifies the **first element** of the local array E. This subroutine computes the location of the first element of the local subarray used, based on *ia*, *desc\_a*, and *p*; therefore, the leading LOCp(*ia+n-1*) part of the local array E contains the local pieces of the leading *ia+n-1* part of the global vector.

Scope: **local**

Specified as: a one-dimensional array of (at least) length LOCp(*ia+n-1*), containing numbers of the data type indicated in Table 110 on page 591. Details about block-cyclic data distribution of global matrix *A* are stored in *desc\_a*.

*ia* is the row or column index of the global matrix *A*, identifying the first row or column of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer, where:

- If (the process grid is  $p \times 1$  and DTYPE\_A = 1) or DTYPE\_A = 502,  $1 \leq ia \leq M_A$  and  $ia+n-1 \leq M_A$ .
- If (the process grid is  $1 \times p$  and DTYPE\_A = 1) or DTYPE\_A = 501,  $1 \leq ia \leq N_A$  and  $ia+n-1 \leq N_A$ .

*desc\_a* is the array descriptor for global matrix *A*. Because vectors are one-dimensional data structures, you may use a type-502, type-501, or type-1 array descriptor regardless of whether the process grid is  $p \times 1$  or  $1 \times p$ . For a type-502 array descriptor, the process grid is used as if it is a  $p \times 1$  process grid. For a type-501 array descriptor, the process grid is used as if it is a  $1 \times p$  process grid. For a type-1 array descriptor, the process grid is used as if it is either a  $p \times 1$  process grid or a  $1 \times p$  process grid. The following tables describe three types of array descriptors. For rules on using array descriptors, see “Notes and Coding Rules” on page 597.

Table 111. Type-502 Array Descriptor

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor Type	DTYPE_A=502 for $p \times 1$ or $1 \times p$  where <i>p</i> is the number of processes in a process grid.	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global

Table 111. Type-502 Array Descriptor (continued)

<i>desc_a</i>	Name	Description	Limits	Scope
3	M_A	Number of rows in the global matrix	If $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	MB_A	Row block size	$MB\_A \geq 1$ and $0 \leq n \leq (MB\_A)(p) - \text{mod}(ia-1, MB\_A)$	Global
5	RSRC_A	The process row over which the first row of the global matrix is distributed	$0 \geq \text{RSRC\_A} < p$	Global
6	—	Not used by this subroutine.	—	—
7	—	Reserved	—	—

Specified as: an array of (at least) length 7, containing fullword integers.

Table 112. Type-1 Array Descriptor ( $p \times 1$  Process Grid)

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor Type	$DTYPE\_A = 1$ for $p \times 1$  where $p$ is the number of processes in a process grid.	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	$N\_A = 1$	
5	MB_A	Row block size	$MB\_A \geq 1$ and $0 \leq n \leq (MB\_A)(p) - \text{mod}(ia-1, MB\_A)$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global
7	RSRC_A	The process row over which the first row of the global matrix is distributed	$0 \leq \text{RSRC\_A} < p$	Global
8	CSRC_A	The process column over which the first column of the global matrix is distributed	$\text{CSRC\_A} = 0$	Global
9	—	Not used by this subroutine.	—	—

Specified as: an array of (at least) length 9, containing fullword integers.

Table 113. Type-501 Array Descriptor

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor Type	$DTYPE\_A=501$ for $1 \times p$ or $p \times 1$  where $p$ is the number of processes in a process grid.	Global

Table 113. Type-501 Array Descriptor (continued)

<i>desc_a</i>	Name	Description	Limits	Scope
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	N_A	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
4	NB_A	Column block size	$NB\_A \geq 1$ and $0 \leq n \leq (NB\_A)(p) - \text{mod}(ia-1, NB\_A)$	Global
5	CSRC_A	The process column over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < p$	Global
6	—	Not used by this subroutine.	—	—
7	—	Reserved	—	—

Specified as: an array of (at least) length 7, containing fullword integers.

Table 114. Type-1 Array Descriptor ( $1 \times p$  Process Grid)

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	$DTYPE\_A = 1$ for $1 \times p$  where $p$ is the number of processes in a process grid.	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	$M\_A = 1$	Global
4	N_A	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$ and $0 \leq n \leq (NB\_A)(p) - \text{mod}(ia-1, NB\_A)$	Global
7	RSRC_A	The process row over which the first row of the global matrix is distributed	$RSRC\_A = 0$	Global
8	CSRC_A	The process column over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < p$	Global
9	—	Not used by this subroutine.	—	—

Specified as: an array of (at least) length 9, containing fullword integers.

- b* is the local part of the global general matrix  $B$ , containing the multiple right-hand sides of the system. This identifies the **first element** of the local array  $B$ . This subroutine computes the location of the first element of the local subarray used, based on  $ib$ ,  $desc\_b$ , and  $p$ ; therefore, the leading

$\text{LOCp}(ib+n-1)$  by  $nrhs$  part of the local array  $B$  must contain the local pieces of the leading  $ib+n-1$  by  $nrhs$  part of the global matrix.

Scope: **local**

Specified as: an  $\text{LLD}_B$  by (at least)  $nrhs$  array, containing numbers of the data type indicated in Table 110 on page 591. Details about the block-cyclic data distribution of global matrix  $B$  are stored in  $desc\_b$ .

$ib$  is the row index of the global matrix  $B$ , identifying the first row of the submatrix  $B$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ib \leq M_B$  and  $ib+n-1 \leq M_B$ .

$desc\_b$  is the array descriptor for global matrix  $B$ , which may be type 502 or type 1, as described in the following tables. For type-502 array descriptor, the process grid is used as if it is a  $p \times 1$  process grid. For rules on using array descriptors, see “Notes and Coding Rules” on page 597.

$desc\_b$	Name	Description	Limits	Scope
1	DTYPE_B	Descriptor type	$\text{DTYPE}_B = 502$ for $p \times 1$ or $1 \times p$ where $p$ is the number of processes in a process grid.	Global
2	CTXT_B	BLACS context	Valid value, as returned by $\text{BLACS\_GRIDINIT}$ or $\text{BLACS\_GRIDMAP}$	Global
3	$M_B$	Number of rows in the global matrix	If $n = 0$ : $M_B \geq 0$ Otherwise: $M_B \geq 1$	Global
4	$MB_B$	Row block size	$MB_B \geq 1$ and $0 \leq n \leq (MB_B)(p) - \text{mod}(ib-1, MB_B)$	Global
5	$\text{RSRC}_B$	The process row over which the first row of the global matrix is distributed	$0 \leq \text{RSRC}_B < p$	Global
6	$\text{LLD}_B$	Leading dimension	$\text{LLD}_B \geq \max(1, \text{LOCp}(M_B))$	<b>Local</b>
7	—	Reserved	—	—

Specified as: an array of (at least) length 7, containing fullword integers.

$desc\_b$	Name	Description	Limits	Scope
1	DTYPE_B	Descriptor type	$\text{DTYPE}_B = 1$ for $p \times 1$ where $p$ is the number of processes in a process grid.	Global
2	CTXT_B	BLACS context	Valid value, as returned by $\text{BLACS\_GRIDINIT}$ or $\text{BLACS\_GRIDMAP}$	Global
3	$M_B$	Number of rows in the global matrix	If $n = 0$ : $M_B \geq 0$ Otherwise: $M_B \geq 1$	Global
4	$N_B$	Number of columns in the global matrix	$N_B \geq nrhs$	Global

<i>desc_b</i>	Name	Description	Limits	Scope
5	MB_B	Row block size	$MB\_B \geq 1$ and $0 \leq n \leq (MB\_B)(p) - \text{mod}(ib-1, MB\_B)$	Global
6	NB_B	Column block size	$NB\_B \geq 1$	Global
7	RSRC_B	The process row over which the first row of the global matrix is distributed	$0 \leq RSRC\_B < p$	Global
8	CSRC_B	The process column over which the first column of the global matrix is distributed	$CSRC\_B = 0$	Global
9	LLD_B	Leading dimension	$LLD\_B \geq \max(1, \text{LOCp}(M\_B))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

*af* is a work area used by this subroutine and contains part of the factorization produced on a preceding call to PDPTTRF. Its size is specified by *laf*.

Scope: **local**

Specified as: a one-dimensional array of (at least) length *laf*, containing numbers of the data type indicated in Table 110 on page 591.

*laf* is the number of elements in array AF.

Scope: **local**

Specified as: a fullword integer, where:

- If (the process grid is  $p \times 1$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 502$ ,  
 $laf \geq 12P+3(MB\_A)$ .
- If (the process grid is  $1 \times p$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 501$ ,  
 $laf \geq 12P+3(NB\_A)$ .

where, in the above formulas, P is the **actual** number of processes containing data.

*work* has the following meaning:

If  $lwork = 0$ , *work* is ignored.

If  $lwork \neq 0$ , *work* is the work area used by this subroutine, where:

- If  $lwork \neq -1$ , the size of *work* is (at least) of length *lwork*.
- If  $lwork = -1$ , the size of *work* is (at least) of length 1.

Scope: **local**

Specified as: an area of storage containing numbers of data type indicated in Table 110 on page 591.

*lwork* is the number of elements in array WORK.

Scope:

- If  $lwork \geq 0$ , *lwork* is **local**
- If  $lwork = -1$ , *lwork* is **global**

Specified as: a fullword integer; where:

- If  $lwork = 0$ , this subroutine dynamically allocates the work area used by the subroutine. The work area is deallocated before control is returned to the calling program. This option is an extension to the ScaLAPACK standard.
- If  $lwork = -1$ , PDPTTRS performs a work area query and return the optimum size of  $work$  in  $work_1$ . No computation is performed and the subroutine returns after error checking is complete.
- Otherwise,  $lwork \geq (10+2\min(100,nrhs))P+4(nrhs)$ , where  $P$  is the **actual** number of processes containing data.

*info* See On Return.

### On Return

*b*  $b$  is the updated local part of the global matrix  $B$ , containing the solution vectors.

Scope: **local**

Returned as: an LLD\_B by (at least)  $nrhs$  array, containing numbers of the data type indicated in Table 110 on page 591.

*work* is the work area used by this subroutine if  $lwork \neq 0$ , where:

If  $lwork \neq 0$  and  $lwork \neq -1$ , the size of  $work$  is (at least) of length  $lwork$ .

If  $lwork = -1$ , the size of  $work$  is (at least) of length 1.

Scope: **local**

Returned as: an area of storage, containing numbers of data type indicated in Table 110 on page 591, where:

- If  $lwork = -1$ , the  $work_1$  is set to the optimum  $lwork$  value needed.
- If  $lwork \geq 1$ , the  $work_1$  is set to the minimum  $lwork$  value needed.

Except for  $work_1$ , the contents of  $work$  are overwritten on return.

*info* indicates that a successful computation or work area query occurred.

Scope: **global**

Returned as: a fullword integer;  $info = 0$ .

## Notes and Coding Rules

1. In your C program, argument *info* must be passed by reference.
2. The output from the PDPTTRF subroutine should be used only as input to the solve subroutine PDPTTRS.

The factored matrix  $A$  is stored in an internal format that depends on the number of processes.

The scalar data specified for input argument  $n$  must be the same for both PDPTTRF and PDPTTRS.

The global vectors for  $d$ ,  $e$ , and  $af$  input to PDPTTRS must be the same as the corresponding output arguments for PDPTTRF; and thus, the scalar data specified for  $ia$ ,  $desc_a$ , and  $laf$  must also be the same.

3. In all cases, follow these rules:
  - $ia = ib$
  - $CTXT\_A = CTXT\_B$
  - If (the process grid is  $p \times 1$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 502$ ,  $MB\_A = MB\_B$ .

- If (the process grid is  $1 \times p$  and  $\text{DTYPE\_A} = 1$ ) or  $\text{DTYPE\_A} = 501$ ,  $\text{NB\_A} = \text{MB\_B}$ .
- If  $\text{DTYPE\_A}=1$ , then:
  - For a  $p \times 1$  process grid (where  $p>1$ ),  $\text{N\_A}=1$ ,  $\text{NB\_A}\geq 1$ , and  $\text{CSRC\_A}=0$ .
  - For a  $1 \times p$  process grid (where  $p>1$ ),  $\text{M\_A}=1$ ,  $\text{MB\_A}\geq 1$ , and  $\text{RSRC\_A}=0$ .
  - For a  $1 \times 1$  process grid:
    - If  $\text{N\_A}=1$ ,  $\text{NB\_A}\geq 1$  and  $\text{CSRC\_A}=0$ .
    - If  $\text{M\_A}=1$ ,  $\text{MB\_A}\geq 1$  and  $\text{RSRC\_A}=0$ .
- If  $\text{DTYPE\_B}=1$ ,  $\text{N\_B}\geq \text{nrhs}$ ,  $\text{NB\_B}\geq 1$ , and  $\text{CSRC\_B}=0$ .
- Following are the consistent combinations of array descriptor types and process grids, where  $p$  is the number of processes in the process grid:

DTYPE_A	DTYPE_B	Process Grid
501	502	$p \times 1$ or $1 \times p$
502	502	$p \times 1$ or $1 \times p$
501	1	$p \times 1$
502	1	$p \times 1$
1	502	$p \times 1$ or $1 \times p$
1	1	$p \times 1$

- To determine the values of  $\text{LOCp}(n)$  used in the argument descriptions, see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28 for descriptor type-1 or “Determining the Number of Rows or Columns in Your Local Arrays” on page 32 for descriptor type-501 and type-502.
- $d$ ,  $e$ ,  $af$  and  $work$  must have no common elements; otherwise, results are unpredictable.
- The global positive definite symmetric tridiagonal matrix  $A$  must be stored in parallel-symmetric-tridiagonal storage mode and distributed over a one-dimensional process grid, using block-cyclic data distribution. See the section on block-cyclically distributing a tridiagonal matrix in “Matrices” on page 40.  
For more information on using block-cyclic data distribution, see “Specifying Block-Cyclically-Distributed Matrices for the Banded Linear Algebraic Equations” on page 29.
- Matrix  $B$  must be distributed over a one-dimensional process grid, using block-cyclic data distribution. For more information using block-cyclic data distribution, see “Specifying Block-Cyclically-Distributed Matrices for the Banded Linear Algebraic Equations” on page 29. Also, see the section on distributing the right-hand side matrix in “Matrices” on page 40.
- If  $lwork = -1$  on any process, it must equal  $-1$  on all processes. That is, if a subset of the processes specifies  $-1$  for the work area size, they must all specify  $-1$ .
- Although global matrices  $A$  and  $B$  may be block-cyclically distributed on a  $1 \times p$  or  $p \times 1$  process grid, the values of  $n$ ,  $ia$ ,  $ib$ ,  $\text{MB\_A}$  (if (the process grid is  $p \times 1$  and  $\text{DTYPE\_A} = 1$ ) or  $\text{DTYPE\_A} = 502$ ),  $\text{NB\_A}$  (if (the process grid is  $1 \times p$  and  $\text{DTYPE\_A} = 1$ ) or  $\text{DTYPE\_A} = 501$ ), must be chosen so that each process has at most one full or partial block of each of the global submatrices  $A$  and  $B$ .
- For global tridiagonal matrix  $A$ , use of the type-1 array descriptor is an extension to ScaLAPACK 1.5. If your application needs to run with both Parallel ESSL and ScaLAPACK 1.5, it is suggested that you use either a type-501 or a type-502 array descriptor for the matrix  $A$ .



## Error Conditions

### Computational Errors

None

**Note:** If the factorization performed by PDPTTRF failed because of a nonpositive definite matrix  $A$ , the results returned by this subroutine are unpredictable. For details, see the *info* output argument for PDPTTRF.

### Resource Errors

Unable to allocate workspace

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. DTYPE\_A is invalid.
2. DTYPE\_B is invalid.

#### Stage 2:

1. CTXT\_A is invalid.

#### Stage 3:

1. This subroutine was called from outside the process grid.

#### Stage 4:

**Note:** In the following error conditions:

- If  $M\_A = 1$  and  $DTYPE\_A = 1$ , a  $1 \times 1$  process grid is treated as a  $1 \times p$  process grid.
  - If  $N\_A = 1$  and  $DTYPE\_A = 1$ , a  $1 \times 1$  process grid is treated as a  $p \times 1$  process grid.
1. The process grid is not  $1 \times p$  or  $p \times 1$ .
  2.  $CTXT\_A \neq CTXT\_B$
  3.  $n < 0$
  4.  $ia < 1$
  5.  $DTYPE\_A = 1$  and  $M\_A \neq 1$  and  $N\_A \neq 1$   
If (the process grid is  $1 \times p$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 501$ :
  6.  $N\_A < 0$  and ( $n = 0$ );  $N\_A < 1$  otherwise
  7.  $NB\_A < 1$
  8.  $n > (NB\_A)(p) - \text{mod}(ia-1, NB\_A)$
  9.  $ia > N\_A$  and ( $n > 0$ )
  10.  $ia+n-1 > N\_A$  and ( $n > 0$ )
  11.  $CSRC\_A < 0$  or  $CSRC\_A \geq p$
  12.  $NB\_A \neq MB\_B$
  13.  $CSRC\_A \neq RSRC\_B$   
If the process grid is  $1 \times p$  and  $DTYPE\_A = 1$ :
  14.  $M\_A \neq 1$
  15.  $MB\_A < 1$
  16.  $RSRC\_A \neq 0$   
If (the process grid is  $p \times 1$  and  $DTYPE\_A = 1$ ) or  $DTYPE\_A = 502$ :
  17.  $M\_A < 0$  and ( $n = 0$ );  $M\_A < 1$  otherwise
  18.  $MB\_A < 1$
  19.  $n > (MB\_A)(p) - \text{mod}(ia-1, MB\_A)$
  20.  $ia > M\_A$  and ( $n > 0$ )
  21.  $ia+n-1 > M\_A$  and ( $n > 0$ )

22.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
23.  $MB\_A \neq MB\_B$
24.  $RSRC\_A \neq RSRC\_B$   
If the process grid is  $p \times 1$  and  $DTYPE\_A = 1$ :
25.  $N\_A \neq 1$
26.  $NB\_A < 1$
27.  $CSRC\_A \neq 0$   
In all cases:
28.  $ia \neq ib$
29.  $DTYPE\_B = 1$  and the process grid is  $1 \times p$  and  $p > 1$
30.  $nrhs < 0$
31.  $ib < 1$
32.  $M\_B < 0$  and  $(n = 0)$ ;  $M\_B < 1$  otherwise
33.  $MB\_B < 1$
34.  $ib > M\_B$  and  $(n > 0)$
35.  $ib+n-1 > M\_B$  and  $(n > 0)$
36.  $RSRC\_B < 0$  or  $RSRC\_B \geq p$
37.  $LLD\_B < \max(1, LOCp(M\_B))$   
If  $DTYPE\_B = 1$ :
38.  $N\_B < 0$  and  $(nrhs = 0)$ ;  $N\_B < 1$  otherwise
39.  $N\_B < nrhs$
40.  $NB\_B < 1$
41.  $CSRC\_B \neq 0$   
In all cases:
42.  $laf < (\text{minimum value})$  (For the minimum value, see the *laf* argument description.)
43.  $lwork \neq 0$ ,  $lwork \neq -1$ , and  $lwork < (\text{minimum value})$  (For the minimum value, see the *lwork* argument description.)

#### Stage 5:

Each of the following global input arguments are checked to determine whether its value is the same on all processes in the process grid:

1.  $n$  differs.
2.  $nrhs$  differs.
3.  $ia$  differs.
4.  $ib$  differs.
5.  $DTYPE\_A$  differs.  
If  $DTYPE\_A = 1$  on all processes:
6.  $M\_A$  differs.
7.  $N\_A$  differs.
8.  $MB\_A$  differs.
9.  $NB\_A$  differs.
10.  $RSRC\_A$  differs.
11.  $CSRC\_A$  differs.  
If  $DTYPE\_A = 501$  on all processes:
12.  $N\_A$  differs.
13.  $NB\_A$  differs.
14.  $CSRC\_A$  differs.  
If  $DTYPE\_A = 502$  on all processes:
15.  $M\_A$  differs.
16.  $MB\_A$  differs.
17.  $RSRC\_A$  differs.  
In all cases:

18. DTYPE\_B differs.  
If DTYPE\_B = 1 on all processes:
19. M\_B differs.
20. N\_B differs.
21. MB\_B differs.
22. NB\_B differs.
23. RSRC\_B differs.
24. CSRC\_B differs.  
If DTYPE\_B = 502 on all processes:
25. M\_B differs.
26. MB\_B differs.
27. RSRC\_B differs.
- Also:
28. *lwork* = -1 on a subset of processes.

## Examples

### Example

This example shows how to solve the system  $AX=B$ , where matrix  $A$  is the same positive definite symmetric tridiagonal matrix factored in “Example” on page 587 for PDPTTRF.

#### Notes:

1. The vectors  $d$  and  $e$ , output from PDPTTRF, are stored in an internal format that depends on the number of processes. These vectors are passed, unchanged, to the solve subroutine PDPTTRS.
2. The contents of the *af* vector, output from PDPTTRF, is not shown. This vector is passed, unchanged, to the solve subroutine PDPTTRS.
3. Because *lwork* = 0, this subroutine dynamically allocates the work area used by this subroutine.

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 3
NPCOL = 1
CALL BLACS_GET (0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      N NRHS D E IA  DESC_A B IB  DESC_B AF  LAF WORK LWORK INFO
CALL PDPTTRS( 12 , 3 , D, E , 1 , DESC_A , B , 1 , DESC_B, AF , 48 , WORK , 0 , INFO)
```

	Desc_A
DTYPE_	502
CTXT_	<i>icontxt</i> <sup>1</sup>
M_	12
MB_	4
RSRC_	0
Not used	—
Reserved	—

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.

	Desc_B
DTYPE_	502
CTXT_	<i>icontxt</i> <sup>1</sup>
M_	12
MB_	4
RSRC_	0
LLD_B	4
Reserved	—

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.

Global vector  $d$  with block size of 4:

B,D	0
	$\begin{bmatrix} .25 \\ .25 \\ .25 \\ 4.0 \\ \text{----} \\ .2 \\ .24 \\ .25 \\ 4.01 \\ \text{----} \\ 4.01 \\ .25 \\ .24 \\ .2 \end{bmatrix}$
0	
1	
2	

Global vector  $e$  with block size of 4:

B,D	0
	$\begin{bmatrix} 2.0 \\ 2.0 \\ 2.0 \\ 2.0 \\ \text{----} \\ 2.0 \\ 2.0 \\ 2.0 \\ 2.0 \\ \text{----} \\ .49 \\ .48 \\ .4 \\ . \end{bmatrix}$
0	
1	
2	

The following is the  $3 \times 1$  process grid:

B,D	0
0	P <sub>00</sub>

1	$P_{10}$
2	$P_{20}$

Local array D with block size of 4:

p,q	0
	.25
	.25
0	.25
	4.0
	.2
	.24
1	.25
	4.01
	4.01
	.25
2	.24
	.2

Local array E with block size of 4:

p,q	0
	2.0
	2.0
0	2.0
	2.0
	2.0
	2.0
1	2.0
	2.0
	2.0
	2.0
	.49
	.48
2	.4
	.

Global matrix  $B$  with a block size of 4:

p,q	0
	70.0 8.0 6.0
	99.0 18.0 9.0
0	90.0 27.0 9.0
	81.0 36.0 9.0
	72.0 45.0 9.0
	63.0 54.0 9.0
1	54.0 63.0 9.0
	45.0 72.0 9.0
	36.0 81.0 9.0
	27.0 90.0 9.0
2	18.0 99.0 9.0
	9.0 82.0 7.0

The following is the  $3 \times 1$  process grid:

B,D	0
0	$P_{00}$

1	$P_{10}$
2	$P_{20}$

Local matrix  $B$  with block size of 4:

p,q	0
0	70.0 8.0 6.0 99.0 18.0 9.0 90.0 27.0 9.0 81.0 36.0 9.0
1	72.0 45.0 9.0 63.0 54.0 9.0 54.0 63.0 9.0 45.0 72.0 9.0
2	36.0 81.0 9.0 27.0 90.0 9.0 18.0 99.0 9.0 9.0 82.0 7.0

**Output:**

Global matrix  $B$  with block size of 4:

B,D	0
0	12.0 1.0 1.0 11.0 2.0 1.0 10.0 3.0 1.0 9.0 4.0 1.0
1	8.0 5.0 1.0 7.0 6.0 1.0 6.0 7.0 1.0 5.0 8.0 1.0
2	4.0 9.0 1.0 3.0 10.0 1.0 2.0 11.0 1.0 1.0 12.0 1.0

The following is the  $3 \times 1$  process grid:

B,D	0
0	$P_{00}$
1	$P_{10}$
2	$P_{20}$

Local matrix  $B$  with block size of 4:

p,q	0
0	12.0 1.0 1.0 11.0 2.0 1.0 10.0 3.0 1.0 9.0 4.0 1.0
1	8.0 5.0 1.0 7.0 6.0 1.0 6.0 7.0 1.0

	5.0	8.0	1.0
	4.0	9.0	1.0
	3.0	10.0	1.0
2	2.0	11.0	1.0
	1.0	12.0	1.0

The value of *info* is 0 on all processes.

---

## Fortran 90 Sparse Linear Algebraic Equation Subroutines and Their Utility Subroutines

This section contains the sparse linear algebraic equation subroutine descriptions and their sparse utility subroutines.



## PADALL — Allocates Space for an Array Descriptor for a General Sparse Matrix

### Purpose

This sparse utility subroutine allocates space for an array descriptor, which is needed to establish a mapping between the global general sparse matrix  $A$  and its corresponding distributed memory location. This subroutine also initializes the components of the array descriptor *desc\_a*.

### Syntax

#### On Entry

*n* is the order of the global general sparse matrix  $A$  and the size of the index space.

Scope: **global**

Type: **required**

Specified as: a fullword integer, where:  $n > 0$ .

*parts* is a user-supplied subroutine that specifies a mapping between a global index for an element in the global general sparse matrix  $A$  and its corresponding storage location on one or more processes.

Sample *parts* subroutines for common types of data distributions are shown in “Sample PARTS Subroutine” on page 956.

For details about how you must define the PARTS subroutine, see “Programming Considerations for the Parts Subroutine (Fortran 90 and Fortran 77)” on page 62.

Scope: **global**

Type: **required**

Specified as: *parts* must be declared as an external subroutine in your application program. It can be whatever name you choose.

*desc\_a* See On Return.

*icontxt* is the BLACS context parameter.

Scope: **global**

Type: **required**

Specified as: a fullword integer that was returned in a prior call to BLACS\_GRIDINIT or BLACS\_GRIDMAP.

#### On Return

*desc\_a* is the local space allocated for the array descriptor for the global general sparse matrix  $A$ . This subroutine also initializes the components of the array descriptor *desc\_a*. The components of *desc\_a* are updated with subsequent calls to PSPINS and finalized with a call to PSPASB.

Table 30 on page 59 describes some of the elements of MATRIX\_DATA, which is one component of the array descriptor, that you may want to reference. However, your application programs should not modify the components of the array descriptor directly. These components should only be updated with calls to PSPINS and PSPASB.

Type: **required**

Returned as: the derived data type DESC\_TYPE.

## Notes and Coding Rules

1. Before you call this subroutine, you must create a  $np \times 1$  process grid, where  $np$  is the number of processes.
2. PADALL allocates *desc\_a* as necessary. Prior to further calls to PADALL with the same *desc\_a*, you must call PADFREE; otherwise, there will be a memory leak.

## Error Conditions

### Computational Errors

None

### Resource Errors

1. Unable to allocate work space.
2. Unable to allocate component(s) of *desc\_a*

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. The BLACS context is invalid.

#### Stage 2:

1. This subroutine was called from outside the process grid.

#### Stage 3:

1. The process grid is not  $np \times 1$ .
2.  $n \leq 0$

#### Stage 4:

1. Each of the following global input arguments are checked to determine whether its value differs from the value specified on process  $P_{00}$ :
  - $n$  differs.

#### Stage 5:

1.  $pv$  or  $nv$ , output from the user-supplied *parts* subroutine, was not valid. For valid values, see the appropriate argument description in "Programming Considerations for the Parts Subroutine (Fortran 90 and Fortran 77)" on page 62.

## PSPALL — Allocates Space for a General Sparse Matrix

### Purpose

This sparse utility subroutine allocates space for the local data of a general sparse matrix *A*. It also initializes some values, which are only for internal use, of the general sparse matrix *A*.

### Syntax

Fortran	CALL PSPALL ( <i>a</i> , <i>desc_a</i> ) CALL PSPALL ( <i>a</i> , <i>desc_a</i> , <i>nnz</i> )
---------	---

### On Entry

*a* See On Return.

*desc\_a* is the array descriptor for a global general sparse matrix *A* that is produced on a preceding call to PADALL.

Type: **required**

Specified as: the derived data type DESC\_TYPE.

*nnz* is an estimate of the number of non-zero elements in the local part of the global general sparse matrix *A*. If the actual number of non-zero elements is greater than *nnz*, Parallel ESSL attempts to allocate additional space.

If *nnz* is not present, Parallel ESSL estimates how many non-zero elements, *nnz*, are present based on the order of the global general sparse matrix *A*.

Scope: **local**

Type: **optional**

Specified as: a fullword integer, where *nnz* > 0.

### On Return

*a* is the local space, which contains some internal values that are initialized by Parallel ESSL, allocated for the global general sparse matrix *A*.

Scope: **local**

Type: **required**

Returned as: the derived data type D\_SPMAT.

### Notes and Coding Rules

1. Before you call this subroutine, you must have called PADALL.
2. For details about some of the elements stored in DESC\_A%MATRIX\_DATA, see “Derived Data Type DESC\_TYPE” on page 59.
3. PSPALL allocates matrix *A* as necessary. Prior to further calls to PSPALL with the same matrix *A*, you must call PSPFREE; otherwise, there will be a memory leak.

### Error Conditions

#### Computational Errors

None

### Resource Errors

1. Unable to allocate component(s) of  $A$ .

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. *desc\_a* has not been initialized.

#### Stage 2:

1. The BLACS context is invalid.

#### Stage 3:

1. This subroutine was called from outside the process grid.

#### Stage 4:

1. The process grid is not  $np \times 1$ .
2. *desc\_a* component(s) are not valid.
3.  $mnz \leq 0$

## PGEALL — Allocates Space for a Dense Vector

### Purpose

This sparse utility subroutine allocates space for a dense vector.

### Syntax

<b>Fortran</b>	CALL PGEALL ( <i>x</i> , <i>desc_a</i> )
----------------	--

### On Entry

*x* See On Return.

*desc\_a* is the array descriptor that is produced on a preceding call to PADALL.

Type: **required**

Specified as: the derived data type DESC\_TYPE.

### On Return

*x* is a pointer to the local space of the dense vector.

Scope: **local**

Type: **required**

Returned as: a pointer to an assumed-shape array with shape (:), containing long-precision real numbers.

## Notes and Coding Rules

1. Before you call this subroutine, you must have called PADALL.
2. You do not need a separate array descriptor for a dense vector because it must conform to the size of matrix *A*. For details about some of the elements stored in DESC\_A%MATRIX\_DATA, see “Derived Data Type DESC\_TYPE” on page 59.
3. This subroutine must be called for:
  - Vector *b* containing the right-hand side.
  - Vector *x* containing the initial guess to the solution.
4. PGEALL allocates the dense vector as necessary. Prior to further calls to PGEALL with the same dense vector, you must call PGEFREE; otherwise, there will be a memory leak.

## Error Conditions

### Computational Errors

None

### Resource Errors

1. Unable to allocate the dense vector.

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. *desc\_a* has not been initialized.

#### Stage 2:

## PGEALL

1. The BLACS context is invalid.

### Stage 3:

1. This subroutine was called from outside the process grid.

### Stage 4:

1. The process grid is not  $np \times 1$ .
2. *desc\_a* component(s) are not valid.

## PSPINS — Inserts Local Data into a General Sparse Matrix

### Purpose

This sparse utility subroutine is used by each process to insert all blocks of data it owns into its local part of the general sparse matrix  $A$ .

### Syntax

Fortran	CALL PSPINS ( <i>a</i> , <i>ia</i> , <i>ja</i> , <i>blk</i> , <i>desc_a</i> )
---------	---

### On Entry

*a* is the local part of the global general sparse matrix  $A$  that is produced on a preceding call to PSPALL or previous call(s) to this subroutine.

Scope: **local**

Type: **required**

Specified as: the derived data type D\_SPMAT.

*ia* is the first global row index of the general sparse matrix  $A$  that receives data from the submatrix  $BLCK$ .

Scope: **local**

Type: **required**

Specified as: a fullword integer;  $1 \leq ia \leq DESC\_A\%MATRIX\_DATA(M)$ .

*ja* is the first global column index of the general sparse matrix  $A$  that receives data from the submatrix  $BLCK$ .

Scope: **local**

Type: **required**

Specified as: a fullword integer, where:  $ja = 1$ .

*blk* is the local part of the submatrix  $BLCK$  to be inserted into the global general sparse matrix  $A$ . Each call to this subroutine inserts one contiguous block of rows into the local part of the sparse matrix corresponding to the global

submatrix  $A_{ia:ia+BLCK\%M-1, ja:ja+BLCK\%N-1}$ . This subroutine only can insert blocks of data it owns into its local part of the general sparse matrix  $A$ .  $BLCK$  contains the following components:

- $BLCK\%M$  is the number of local rows in the submatrix  $BLCK$ . Scope: **local**.

Specified as: a fullword integer;

$1 \leq BLCK\%M \leq DESC\_A\%MATRIX\_DATA(N\_ROW)$ .

- $BLCK\%N$  is an upper bound on the number of local columns in the submatrix  $BLCK$ . Scope: **local**.

Specified as: a fullword integer;  $1 \leq BLCK\%N \leq n$ , where  $n$  is the order of the global general sparse matrix  $A$ .

- $BLCK\%FIDA$  is the storage mode for the submatrix  $BLCK$ , where:

If  $BLCK\%FIDA='CSR'$ , the submatrix  $BLCK$  is stored in the storage-by-rows storage mode. Scope: **global**.

Specified as: a character variable of length 5;  $BLCK\%FIDA='CSR'$ .

If `BLCK%FIDA='CSR'`, then you must specify the `BLCK%AS`, `BLCK%IA1`, and `BLCK%IA2` components, as follows:

- `BLCK%AS` is a pointer to the submatrix *BLCK* that is stored by rows. See “Notes and Coding Rules.” Scope: **local**.  
Specified as: a pointer to an assumed-shape array with shape (:), containing long-precision real numbers.
- `BLCK%IA1` is a pointer to the column numbers of each non-zero element in the submatrix *BLCK*. See “Notes and Coding Rules.” Scope: **local**.  
Specified as: a pointer to an assumed-shape array with shape (:), containing fullword integers;  $1 \leq \text{BLCK\%IA1}(i) \leq \text{BLCK\%N}$ , where:  $i = 1, nz$  and  $nz$  is the **actual** number of non-zero elements in the submatrix *BLCK*.
- `BLCK%IA2` is a pointer to the starting positions of each row of the submatrix *BLCK* in `BLCK%AS` and one position past the end of `BLCK%AS`. See “Notes and Coding Rules.” Scope: **local**.  
Specified as: a pointer to an assumed-shape array with shape (:), containing fullword integers, where:
  - `BLCK%IA2(1) = 1`
  - `BLCK%IA2(BLCK%M+1) = 1+nz` and  $nz$  is the **actual** number of non-zero elements in the submatrix *BLCK*.

Specified as: the derived data type `D_SPMAT`.

*desc\_a* is the descriptor vector for a global general sparse matrix *A* that is produced on a preceding call to `PADALL` or previous call(s) to this subroutine.

Type: **required**

Specified as: the derived data type `DESC_TYPE`.

## On Return

*a* is the updated local part of the global general sparse matrix *A*, updated with data from the submatrix *BLCK*.

Scope: **local**

Type: **required**

Returned as: the derived data type `D_SPMAT`.

*desc\_a* is the updated array descriptor for the global general sparse matrix *A*.

Type: **required**

Returned as: the derived data type `DESC_TYPE`.

## Notes and Coding Rules

1. Before you call this subroutine, you must have called `PADALL` and `PSPALL`.
2. This subroutine accepts mixed case letters for the `BLCK%FIDA` component.
3. Arguments *BLCK* and *A* must not have common elements; otherwise, results are unpredictable.
4. For details about some of the elements stored in `DESC_A%MATRIX_DATA`, see “Derived Data Type `DESC_TYPE`” on page 59.
5. The submatrix *BLCK* must be stored by rows; that is `BLCK%FIDA = 'CSR'`. For information about the storage-by-rows storage mode, see the *ESSL Guide and Reference*.



6. Once you declare `BLCK` of derived data type `D_SPMAT`, you must allocate the components of `BLCK` that point to an array. The following example shows how to code the allocate statement if each row of the submatrix *BLCK* contains no more than 20 elements:

```
TYPE(D_SPMAT) :: BLCK                                !Declare the BLCK variable
```

```
      .
      .
      .
```

```
ALLOCATE(BLCK%AS(20),BLCK%IA1(20),BLCK%IA2(2)) !Allocate array pointers
```

When you are finished calling PSPINS, you should deallocate `BLCK%AS`, `BLCK%IA1`, and `BLCK%IA2`.

7. Each process has to call PSPINS as many times as necessary to insert the local rows it owns. It is also possible to call PSPINS multiple times to insert different or duplicate coefficients of the same local row it owns. For information on how duplicate coefficients are handled, see the *dupflag* argument description in PSPASB. For an example of inserting coefficients of the same local row, see “Example.”

## Error Conditions

### Computational Errors

None

### Resource Errors

1. Unable to allocate work space.
2. Unable to allocate component(s) of *A*.

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. *desc\_a* has not been initialized.

#### Stage 2:

1. The BLACS context is invalid.

#### Stage 3:

1. This subroutine was called from outside the process grid.

#### Stage 4:

1. The process grid is not  $np \times 1$ .
2.  $ia < 1$  or  $ia > \text{DESC\_A\%MATRIX\_DATA}(M)$
3.  $ja \neq 1$
4. *desc\_a* component(s) are not valid.
5. The sparse matrix *A* is not valid.
6.  $\text{BLCK\%M} < 1$  or  $\text{BLCK\%M} > \text{DESC\_A\%MATRIX\_DATA}(N\_ROW)$
7.  $\text{BLCK\%N} < 1$  or  $\text{BLCK\%N} > n$
8.  $\text{BLCK\%FIDA} \neq \text{'CSR'}$
9. One or more rows to be inserted into submatrix *A* does not belong to the process.

## Examples

### Example

This piece of an example shows how to insert coefficients into the same `GLOB_ROW` row by calling PSPINS multiple times. It would be useful in finite element applications, where PSPINS inserts one element at a time into the global

matrix, but more than one element may contribute to the same matrix row. In this case, PSPINS is called with the same value of *ia* by all the elements contributing to that row.

For a complete example, see “Example—Using the Fortran 90 Sparse Subroutines” on page 636.

```

      .
      .
      .
DO GLOB_ROW = 1, N

  ROW_MAT%DESCRA(1) = 'G'
  ROW_MAT%FIDA      = 'CSR'

  ROW_MAT%IA2(1) = 1
  ROW_MAT%IA2(2) = 1

  IA = GLOB_ROW

  !      (x-1,y,z)
  ROW_MAT%AS(1) = COEFF(X-1,Y,Z,X,Y,Z)
  ROW_MAT%IA1(1) = IDX(X-1,Y,Z)
  CALL PSPINS(A,IA,1,ROW_MAT,DESC_A)
  !      (x,y-1,z)
  ROW_MAT%AS(1) = COEFF(X,Y-1,Z,X,Y,Z)
  ROW_MAT%IA1(1) = IDX(X,Y-1,Z)
  CALL PSPINS(A,IA,1,ROW_MAT,DESC_A)
  !      (x,y,z-1)
  ROW_MAT%AS(1) = COEFF(X,Y,Z-1,X,Y,Z)
  ROW_MAT%IA1(1) = IDX(X,Y,Z-1)
  CALL PSPINS(A,IA,1,ROW_MAT,DESC_A)
  !      (x,y,z)
  ROW_MAT%AS(1) = COEFF(X,Y,Z,X,Y,Z)
  ROW_MAT%IA1(1) = IDX(X,Y,Z)
  CALL PSPINS(A,IA,1,ROW_MAT,DESC_A)
  !      (x,y,z+1)
  ROW_MAT%AS(1) = COEFF(X,Y,Z+1,X,Y,Z)
  ROW_MAT%IA1(1) = IDX(X,Y,Z+1)
  CALL PSPINS(A,IA,1,ROW_MAT,DESC_A)
  !      (x,y+1,z)
  ROW_MAT%AS(1) = COEFF(X,Y+1,Z,X,Y,Z)
  ROW_MAT%IA1(1) = IDX(X,Y+1,Z)
  CALL PSPINS(A,IA,1,ROW_MAT,DESC_A)
  !      (x+1,y,z)
  ROW_MAT%AS(1) = COEFF(X+1,Y,Z,X,Y,Z)
  ROW_MAT%IA1(1) = IDX(X+1,Y,Z)
  CALL PSPINS(A,IA,1,ROW_MAT,DESC_A)
END DO

      .
      .
      .

```

## PGEINS — Inserts Local Data into a Dense Vector

### Purpose

This sparse utility subroutine is used by each process to insert all blocks of data it owns into its local part of the dense vector.

### Syntax

<b>Fortran</b>	CALL PGEINS ( <i>x</i> , <i>blk</i> , <i>desc_a</i> , <i>ix</i> )
----------------	---

### On Entry

- x* is a pointer to the local space for the dense vector that is produced by a preceding call to PGEALL or previous call(s) to this subroutine.  
 Scope: **local**  
 Type: **required**  
 Specified as: a pointer to an assumed-shape array with shape (:), containing long-precision real numbers.
- blk* is the local part of the submatrix *BLCK* to be inserted into the dense vector. Each call to this subroutine inserts one contiguous block of data into the local part of the dense vector corresponding to the global submatrix  $X_{ix:ix+size(blk,1)-1}$ . This subroutine only inserts a block of data it owns into its local part of the dense vector.  
 Scope: **local**  
 Type: **required**  
 Specified as: an assumed-shape array with shape (:), containing long-precision real numbers, where:  
 $1 \leq size(blk,1) \leq DESC\_A\%MATRIX\_DATA(N\_ROW)$
- desc\_a* is the array descriptor that is produced by a preceding call to PADALL or PSPINS.  
 Type: **required**  
 Specified as: the derived data type DESC\_TYPE.
- ix* is the first global row index of the dense vector that receives data from the submatrix *BLCK*.  
 Scope: **local**  
 Type: **optional**  
 Specified as: a fullword integer;  $1 \leq ix \leq DESC\_A\%MATRIX\_DATA(M)$ .  
 The default value is 1.

### On Return

- x* is a pointer to the local space for the dense vector, updated with local data from the submatrix *BLCK*.  
 Scope: **local**  
 Type: **required**  
 Returned as: a pointer to an assumed-sized array with shape (:), containing long-precision real numbers.

## Notes and Coding Rules

1. Before you call this subroutine, you must have called PGEALL and PADALL.
2. You do not need a separate array descriptor for a dense vector because it must conform to the size of matrix *A*. For details about some of the elements stored in DESC\_A%MATRIX\_DATA, see “Derived Data Type DESC\_TYPE” on page 59.
3. This subroutine must be called for:
  - Vector *b* containing the right-hand side.
  - Vector *x* containing the initial guess to the solution.
4. Each process has to call PGEINS as many times as necessary to insert the local elements it owns. It is also possible to call PGEINS multiple times to insert different coefficients of the same local row it owns. Duplicate coefficients are overwritten.

## Error Conditions

### Computational Errors

None

### Resource Errors

None.

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. *desc\_a* has not been initialized.

#### Stage 2:

1. The BLACS context is invalid.

#### Stage 3:

1. This subroutine was called from outside the process grid.

#### Stage 4:

1. *desc\_a* component(s) are not valid.
2. The process grid is not  $np \times 1$ .
3.  $ix < 1$  or  $ix > \text{DESC\_A\%MATRIX\_DATA}(M)$
4.  $\text{size}(x,1) < \max(1, \text{DESC\_A\%MATRIX\_DATA}(N\_ROW))$
5.  $\text{size}(blk,1) < 1$  or  $\text{size}(blk,1) > \text{DESC\_A\%MATRIX\_DATA}(N\_ROW)$

#### Stage 5:

1. One or more elements to be inserted into the dense vector does not belong to the process.

## PSPASB — Assembles a General Sparse Matrix

### Purpose

This sparse utility subroutine uses the output from PSPINS to assemble the global general sparse matrix  $A$  and its array descriptor  $desc\_a$ .

### Syntax

<b>Fortran</b>	CALL PSPASB ( <i>a</i> , <i>desc_a</i> )
	CALL PSPASB ( <i>a</i> , <i>desc_a</i> , <i>mtype</i> , <i>stor</i> , <i>dupflag</i> , <i>info</i> )

### On Entry

*a* is the local part of the global general sparse matrix  $A$  that is produced by previous call(s) to PSPINS.

Scope: **local**

Type: **required**

Specified as: the derived data type D\_SPMAT.

*desc\_a* is the array descriptor for the global general sparse matrix  $A$  that is produced by previous call(s) to PSPINS.

Type: **required**

Specified as: the derived data type DESC\_TYPE.

*mtype* indicates the form of the global sparse matrix  $A$  used, where:

If *mtype* = 'GEN',  $A$  is a general sparse matrix.

Scope: **global**

Type: **optional**

Specified as: a character variable of length 5; *mtype* = 'GEN'. The default value is 'GEN'.

*stor* indicates the storage mode that the global general sparse matrix  $A$  is returned in, where:

If *stor* = 'DEF', this subroutine chooses an appropriate storage mode, which is an internal format accepted by the preconditioner and solver subroutines, for storing the global general sparse matrix  $A$  on output.

If *stor* = 'CSR', the global general sparse matrix  $A$  is stored in the storage-by-rows storage mode on output.

Scope: **global**

Type: **optional**

Specified as: a character variable of length 5; *stor* = 'DEF' or 'CSR'. The default value is 'DEF'.

*dupflag*

is a flag indicating how to use coefficients that are specified more than once on the same process; that is, duplicate coefficients within the same local part of the matrix  $A$ :

If *dupflag* = 0, this subroutine uses the first of the duplicate coefficients.

If *dupflag* = 1, this subroutine adds all the duplicate coefficients with the same indices.

If *dupflag* = 2, this subroutine raises an error condition indicating that there are unexpected duplicate coefficients.

Scope: **global**

Type: **optional**

Specified as: a fullword integer; *dupflag* = 0, 1, or 2. The default value is 0.

*info* See On Return.

## On Return

*a* is the updated local part of the global general sparse matrix *A*, where:

If *stor* = 'DEF', this subroutine chooses an appropriate storage mode, which is an internal format accepted by the preconditioner and solver subroutines, for storing the global general sparse matrix *A* on output.

If *stor* = 'CSR', the global general sparse matrix *A* is stored in the storage-by-rows storage mode on output.

Scope: **local**

Type: **required**

Returned as: the derived data type D\_SPMAT.

*desc\_a* is the final updated array descriptor for the global general sparse matrix *A*.

Type: **required**

Returned as: the derived data type DESC\_TYPE.

*info* has the following meaning, when *info* is **present**:

If *info* = 0, then no input-argument errors or computational errors occurred. This indicates a normal exit.

**Note:** Because Parallel ESSL terminates the application if input-argument errors occur, the setting of *info* is irrelevant for these errors.

If *info* > 0, then one or more of the following computational errors occurred and the appropriate error messages were issued, indicating an error exit, where:

- If *info* = 1, the sparse matrix *A* contains duplicate coefficients.
- If *info* = 2, the sparse matrix *A* contains empty row(s).

Scope: **global**

Type: **optional**

Returned as: a fullword integer; *info* ≥ 0.

## Notes and Coding Rules

1. Before you call this subroutine, you must have called PSPINS as many times as needed; that is, you must have completed building the matrix with call(s) to PSPINS before you place a call to this subroutine.
2. This subroutine accepts mixed case letters for the *mtype* and *stor* arguments.

3. For details about some of the elements stored in DESC\_A%MATRIX\_DATA, see “Derived Data Type DESC\_TYPE” on page 59.

## Error Conditions

### Computational Errors

The sparse matrix  $A$  contains duplicate coefficients or empty row(s). For details, see the description of the *info* argument.

### Resource Errors

1. Unable to allocate work space.
2. Unable to allocate component(s) of *desc\_a*.
3. Unable to allocate component(s) of  $A$ .

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. *desc\_a* has not been initialized.

#### Stage 2:

1. The BLACS context is invalid.

#### Stage 3:

1. This subroutine was called from outside the process grid.

#### Stage 4:

1. *desc\_a* component(s) are not valid.
2. The process grid is not  $np \times 1$ .
3. The sparse matrix  $A$  is not valid.
4. *mtype*  $\neq$  'GEN'
5. *stor*  $\neq$  'DEF' or 'CSR'
6. *dupflag*  $\neq$  0, 1, or 2
7. Some local rows in the sparse matrix  $A$  are missing.

#### Stage 5:

1. Each of the following global input arguments are checked to determine whether its value differs from the value specified on process  $P_{00}$ :
  - *mtype* differs.
  - *stor* differs.
  - *dupflag* differs.

## PGEASB — Assembles a Dense Vector

### Purpose

This sparse utility subroutine assembles a dense vector.

### Syntax

Fortran	CALL PGEASB ( <i>x</i> , <i>desc_a</i> )
---------	--

### On Entry

*x* is a pointer to the local part of the dense vector that is produced by previous call(s) to PGEINS.

Scope: **local**

Type: **required**

Specified as: a pointer to an assumed-shape array with shape (:), containing long-precision real numbers.

*desc\_a* is the array descriptor, which was finalized in a preceding call to PSPASB.

Type: **required**

Specified as: the derived data type DESC\_TYPE.

### On Return

*x* is a pointer to the local part of the global dense vector.

Scope: **local**

Type: **required**

Returned as: a pointer to an assumed-sized array with shape (:), containing long-precision real numbers.

### Notes and Coding Rules

1. Before you call this subroutine, you must have called PGEINS as many times as needed; that is, you must have completed building the dense vectors with call(s) to PGEINS before you place a call to this subroutine.

Before you call this subroutine, you must have called PSPASB.

2. You do not need a separate array descriptor for a dense vector because it must conform to the size of matrix *A*. For details about some of the elements stored in DESC\_A%MATRIX\_DATA, see “Derived Data Type DESC\_TYPE” on page 59.
3. This subroutine must be called for:
  - Vector *b* containing the right-hand side.
  - Vector *x* containing the initial guess to the solution.

### Error Conditions

#### Computational Errors

None

#### Resource Errors

None.



## Input-Argument and Miscellaneous Errors

### Stage 1:

1. *desc\_a* has not been initialized.

### Stage 2:

1. The BLACS context is invalid.

### Stage 3:

1. This subroutine was called from outside the process grid.

### Stage 4:

1. The process grid is not  $np \times 1$ .
2. *desc\_a* component(s) are not valid.
3.  $\text{size}(x,1) < \text{DESC\_A\%MATRIX\_DATA}(\text{N\_ROW})$

## PSPGPR — Preconditioner for a General Sparse Matrix

### Purpose

This subroutine computes a preconditioner for a global general sparse matrix *A* that should be passed unchanged to the PSPGIS subroutine. The preconditioners include diagonal scaling or an incomplete LU factorization.

### Syntax

Fortran	CALL PSPGPR ( <i>iprec</i> , <i>a</i> , <i>prcs</i> , <i>desc_a</i> ) CALL PSPGPR ( <i>iprec</i> , <i>a</i> , <i>prcs</i> , <i>desc_a</i> , <i>info</i> )
---------	--

### On Entry

*iprec* is a flag that determines the type of preconditioning, where:

If *iprec* = 0, which is referred to as *none*, indicates the local part of the submatrix *A* is not preconditioned. PSPGIS will not be effective in this case, unless the coefficient matrix is well conditioned; if your input matrix is not well conditioned, you should consider using *iprec* = 1 or 2.

If *iprec* = 1, which is referred to as *diagsc*, indicates the local part of the submatrix *A* is preconditioned by a local diagonal submatrix.

If *iprec* = 2, which is referred to as *ilu*, indicates the local part of the submatrix *A* is preconditioned by a local incomplete LU factorization.

It is suggested that you use a preconditioner. For an explanation, see “Notes and Coding Rules” on page 625.

Scope: **global**

Type: **required**

Specified as: a fullword integer, where: *iprec* = 0, 1, or 2.

*a* is the local part of the global general sparse matrix *A*, finalized on a preceding call to PSPASB.

Scope: **local**

Type: **required**

Specified as: the derived data type D\_SPMAT.

*prcs* See On Return.

*desc\_a* is the array descriptor for the global general sparse matrix *A* that was finalized in a call to PSPASB.

Type: **required**

Specified as: the derived data type DESC\_TYPE.

*info* See On Return.

### On Return

*prcs* is the preconditioner data structure *prcs* that must be passed unchanged to PSPGIS.

Scope: **local**

Type: **required**

Returned as: the derived data type D\_PRECN.

*info* has the following meaning, when *info* is **present**:

If *info* = 0, then no input-argument errors or computational errors occurred. This indicates a normal exit.

**Note:** Because Parallel ESSL terminates the application if input-argument errors occur, the setting of *info* is irrelevant for these errors.

If *info* > 0, the value stored in *info* indicates the row index in the global general sparse matrix *A* where the preconditioner failed.

Scope: **global**

Type: **optional**

Returned as: a fullword integer; *info* ≥ 0.

## Notes and Coding Rules

1. Before you call this subroutine, you must have called PSPASB and PGEASB.
2. PSPGPR allocates *prcs*, as necessary. Prior to further calls to PSPGPR with the same *prcs*, you must call PSPFREE; otherwise, there will be a memory leak.
3. For details about some of the elements stored in DESC\_A%MATRIX\_DATA, see “Derived Data Type DESC\_TYPE” on page 59.
4. Parallel ESSL builds the preconditioner, *prcs*, which is specified as derived data type D\_PRECN, and its components. All the components of derived data type D\_PRECN are used for internal use only.
5. The convergence rate of an iterative method as applied to a given system of linear equations depends on the spectral properties of the coefficient matrix of the linear system; therefore it is often convenient to apply a linear transformation to the system such that the solution of the transformed system is the same (in exact arithmetic) as that of the original, but the spectral properties and the convergence behavior are more favorable. Such a transformation is called preconditioning. If a matrix *M* approximates *A*, then:

$$(M^{-1})Ax = (M^{-1})b$$

is a preconditioned system and *M* is called a preconditioner. In practice, the new coefficient matrix  $(M^{-1})A$  is almost never formed explicitly, but rather its action is computed during the application of the iterative method. The effectiveness of the preconditioning operation depends on a trade-off between how well *M* approximates *A* and how costly it is to compute and invert it; no single preconditioner will give best overall performance under all situations. Note finally that it is quite rare for a linear system to behave well enough so as not to require preconditioning; indeed most linear systems originating from the discretization of difficult physical problems require preconditioning to have any convergence at all.

See references [9] and [39].

## Error Conditions

### Computational Errors

1. The preconditioner for the sparse matrix *A* is unstable. For details, see the *info* output argument for this subroutine.

### Resource Errors

1. Unable to allocate work space.
2. Unable to allocate component(s) of *prcs*.

## Input-Argument and Miscellaneous Errors

### Stage 1:

1. *desc\_a* has not been initialized.

### Stage 2:

1. The BLACS context is invalid.

### Stage 3:

1. This subroutine was called from outside the process grid.

### Stage 4:

1. The process grid is not  $np \times 1$ .
2. *desc\_a* component(s) are not valid.
3. *iprec*  $\neq$  0, 1, or 2
4. The storage format for *A* is not supported.

### Stage 5:

1. Each of the following global input arguments are checked to determine whether its value differs from the value specified on process  $P_{00}$ :
  - *iprec* differs.

PSPGIS — Iterative Linear System Solver for a General Sparse Matrix

Purpose

This subroutine solves a general sparse linear system of equations, using an iterative algorithm, with or without preconditioning. The methods include the more smoothly converging variant of the CGS method (Bi-CGSTAB), conjugate gradient squared (CGS), or transpose-free quasi-minimal residual method (TFQMR).

See references [7], [9], [12], and [36].

Syntax

Fortran	CALL PSPGIS ( <i>a</i> , <i>b</i> , <i>x</i> , <i>prcs</i> , <i>desc_a</i> )
	CALL PSPGIS ( <i>a</i> , <i>b</i> , <i>x</i> , <i>prcs</i> , <i>desc_a</i> , <i>iparm</i> , <i>rparm</i> , <i>info</i> )

On Entry

- a*

is the local part of the coefficient matrix *A*, produced on a previous call to PSPASB.  
Scope: **local**  
Type: **required**  
Specified as: the derived data type D\_SPMAT.
- b*

is a pointer to the local part of the global vector *b*, containing the right-hand side of the matrix problem and produced on a previous call to PGEASB.  
Scope: **local**  
Type: **required**  
Specified as: a pointer to an assumed-shape array with shape (:), containing long-precision real numbers.
- x*

is a pointer to the local part of the global vector *x*, containing the initial guess to the solution of the linear system and produced on a previous call to PGEASB.  
Scope: **local**  
Type: **required**  
Specified as: a pointer to an assumed-shape array with shape (:), containing long-precision real numbers.
- prcs*

is the preconditioner data structure *prcs*, produced on a previous call to PSPGPR.  
Scope: **local**  
Type: **required**  
Specified as: the derived data type D\_PRECN.
- desc\_a*

is the array descriptor, produced on a previous call to PSPASB, for the global general sparse matrix *A*.  
Type: **required**

Specified as: the derived data type DESC\_TYPE.

*iparm* is an array of parameters,  $IPARM(i)$ , where:

- $IPARM(1)$  is the flag, referred to as *methd*, used to select the iterative procedure used, where:
  - If *methd* = 1, the more smoothly converging variant of the CGS method, referred to as Bi-CGSTAB, is used.
  - If *methd* = 2, the conjugate gradient squared method, referred to as CGS, is used.
  - If *methd* = 3, the transpose-free quasi-minimal residual method, referred to as TFQMR, is used.
- $IPARM(2)$  is the flag, *istopc*, used to select the stopping criterion used in the computation, where the following items are used in the definitions of the stopping criteria below:
  - $\epsilon$  is the desired relative accuracy and is stored in *eps*
  - $x_j$  is the solution found at the  $j$ -th iteration.
  - $r_j$  and  $r_0$  are the preconditioned residuals obtained at iterations  $j$  and 0, respectively. (The residual at iteration  $j$  is defined as  $b - Ax_j$ )

If *istopc* = 1, the iterative method is stopped when:

  - $\|r_j\|_2 / \|x_j\|_2 < \epsilon$

If *istopc* = 2, the iterative method is stopped when:

  - $\|r_j\|_2 / \|r_0\|_2 < \epsilon$

If *istopc* = 3, the iterative method is stopped when:

  - $\|x_j - x_{j-1}\|_2 / \|x_j\|_2 < \epsilon$

**Note:** Stopping criterion 3 performs poorly with the TFQMR method; therefore, if you specify TFQMR (*methd* = 3), you should not specify stopping criterion 3.
- $IPARM(3)$  is the maximum number of iterations *itmax* allowed.
- $IPARM(4)$ , referred to as *itrace*, has the following meaning:
  - If *itrace* = 0, then *itrace* is ignored.
  - If *itrace* > 0, an informational message about the convergence, which is based on the stopping criterion described in *istopc*, is issued at every *itrace*-th iteration and upon exit.
- $IPARM(5)$ , see On Return.
- $IPARM(6)$  through  $IPARM(20)$  are reserved.

Scope: **global**

Type: **optional**

Default:

- *methd* = 1
- *istopc* = 1
- *itmax* = 500
- *itrace* = 0

Specified as: an array of length 20, containing fullword integers, where:

- *methd* = 1, 2, or 3
- *istopc* = 1, 2, or 3
- *itmax*  $\geq 0$
- *itrace*  $\geq 0$
- $IPARM(6)$  through  $IPARM(20)$  should be set to zero.

*rparm* is an array of parameters,  $\text{RPARM}(i)$ , where:

- $\text{RPARM}(1)$ , referred to as *eps*, is the relative accuracy  $\epsilon$  used in the stopping criterion.
- $\text{RPARM}(2)$ , see On Return.
- $\text{RPARM}(3)$  through  $\text{RPARM}(20)$  are reserved.

Scope: **global**

Type: **optional**

Default:  $\text{eps} = 10^{-8}$

Specified as: an array of length 20, containing long-precision real numbers, where:

- $\text{eps} \geq 0$ .
- $\text{RPARM}(3)$  through  $\text{RPARM}(20)$  should be set to zero.

*info* See On Return.

### On Return

*x* is a pointer to the local part of the solution vector  $x$

Scope: **local**

Type: **required**

Returned as: a pointer to an assumed-shape array of shape (:), containing long-precision real numbers.

*iparm* has the following meaning, when *iparm* is **present**:

$\text{IPARM}(5)$  is the number of iterations, *iter*, performed by this subroutine.

Scope: **global**

Type: **optional**

Returned as: an array of length 20, containing fullword integers, where  $\text{iter} \geq 0$ .

*rparm* has the following meaning, when *rparm* is **present**:

$\text{RPARM}(2)$  contains the estimate of the error, *err*, of the solution, according to the stopping criterion, *istopc*, in use. For details, see the *istopc* argument description.

Scope: **global**

Type: **optional**

Returned as: an array of length 20, containing long-precision real numbers, where  $\text{err} \geq 0$ .

*info* has the following meaning, when *info* is **present**:

If  $\text{info} = 0$ , then no input-argument errors or computational errors occurred. This indicates a normal exit.

**Note:** Because Parallel ESSL terminates the application if input-argument errors occur, the setting of *info* is irrelevant for these errors.

If  $\text{info} > 0$ , then this subroutine exceeded *itmax* iterations without converging. You may want to try the following to get your matrix to converge:

- You can increase the number of iterations and call this subroutine again without making any other changes to your program.
- You can change the requested precision and/or the stopping criterion; your original precision requirement may be too stringent under a given stopping criterion.
- You can use a preconditioner if you were not already doing so, or to change the one you were using. Note also that the efficiency of the preconditioner may depend on the data distribution strategy adopted. See “Notes and Coding Rules” on page 625.

Scope: **global**

Type: **optional**

Returned as: a fullword integer; *info*  $\geq 0$ .

## Notes and Coding Rules

1. Before you call this subroutine, you must have called PSPGPR.
2. For details about some of the elements stored in DESC\_A%MATRIX\_DATA, see “Derived Data Type DESC\_TYPE” on page 59.
3. Parallel ESSL builds the preconditioner, *prcs*, which is specified as derived data type D\_PRECN, and its components. All the components of derived data type D\_PRECN are used for internal use only.

## Error Conditions

### Computational Errors

This subroutine exceeded *itmax* iterations without converging. Vector *x* contains the approximate solution computed at the last iteration.

**Note:** If the preconditioner computed by PSPGPR failed because the sparse matrix *A* is unstable, the results returned by this subroutine are unpredictable. For details, see the *info* output argument for PSPGPR.

You may want to try the following to get your matrix to converge:

- You can increase the number of iterations and call this subroutine again without making any other changes to your program.
- You can change the requested precision and/or the stopping criterion; your original precision requirement may be too stringent under a given stopping criterion.
- You can use a preconditioner if you were not already doing so, or to change the one you were using. Note also that the efficiency of the preconditioner may depend on the data distribution strategy adopted. See “Notes and Coding Rules” on page 625.

### Resource Errors

1. Unable to allocate work space.

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. *desc\_a* has not been initialized.

#### Stage 2:



1. The BLACS context is invalid.

**Stage 3:**

1. This subroutine was called from outside the process grid.

**Stage 4:**

1. The process grid is not  $np \times 1$ .
2. *desc\_a* component(s) are not valid.
3. The sparse matrix *A* is not valid.
4.  $\text{size}(\text{iparm}) < 20$
5.  $\text{size}(\text{rparm}) < 20$
6.  $\text{eps} < 0.0$
7. *methd*  $\neq$  1, 2, or 3
8. *iprec*  $\neq$  0, 1, or 2
9. *istopc*  $\neq$  1, 2, or 3
10.  $\text{itmax} < 0$
11.  $\text{itrace} < 0$
12. The storage format for the sparse matrix *A* is not supported.

**Stage 5:**

1.  $\text{size}(x) < \text{DESC\_A\%MATRIX\_DATA(N\_ROW)}$
2.  $\text{size}(b) < \text{DESC\_A\%MATRIX\_DATA(N\_ROW)}$
3. The preconditioner data structure *prcs* is not valid.

**Stage 6:**

1. Each of the following global input arguments are checked to determine whether its value differs from the value specified on process  $P_{00}$ :
  - *eps* differs.
  - *methd* differs.
  - *istopc* differs.
  - *itmax* differs.
  - *itrace* differs.
  - Component(s) of *prcs* differ.

## PGEFREE — Deallocates Space for a Dense Vector

### Purpose

This sparse utility subroutine deallocates space that is used for a dense vector.

### Syntax

Fortran	CALL PGEFREE ( <i>x</i> , <i>desc_a</i> )
---------	---

### On Entry

*x* is a pointer to the dense vector *x*.

Scope: **local**

Type: **required**

Specified as: a pointer to an assumed-shape array with shape (:), containing long-precision real numbers.

*desc\_a* is the array descriptor for the sparse matrix *A*.

Type: **required**

Specified as: the derived data type DESC\_TYPE.

### Notes and Coding Rules

1. Before you call this subroutine, you must have called PGEALL.
2. You must deallocate *b*, *x*, sparse matrix *A*, and preconditioner data structure *prcs* before you deallocate the array descriptor *desc\_a*.

### Error Conditions

#### Computational Errors

None

#### Resource Errors

None.

#### Input-Argument and Miscellaneous Errors

##### Stage 1:

1. *desc\_a* has not been initialized.

##### Stage 2:

1. The BLACS context is invalid.

##### Stage 3:

1. This subroutine was called from outside the process grid.

##### Stage 4:

1. The process grid is not  $np \times 1$ .
2. The pointer *x* is not associated and therefore cannot be deallocated.

# PSPFREE — Deallocates Space for a General Sparse Matrix

## Purpose

This sparse utility subroutine deallocates space that is used for a global general sparse matrix *A* or a preconditioner data structure *prcs*.

## Syntax

Fortran	CALL PSPFREE ( <i>a</i> , <i>desc_a</i> )
	CALL PSPFREE ( <i>prcs</i> , <i>desc_a</i> )

## On Entry

- a* is the general sparse matrix *A*.  
Scope: **local**  
Type: **required**  
Specified as: the derived data type D\_SPMAT.
- prcs* is the preconditioner data structure *prcs*.  
Scope: **local**  
Type: **required**  
Specified as: the derived data type D\_PRECN.
- desc\_a* is the array descriptor for the sparse matrix *A*.  
Type: **required**  
Specified as: the derived data type DESC\_TYPE.

## Notes and Coding Rules

- Before you call this subroutine to deallocate the sparse matrix *A*, you must have called PSPALL.  
Before you call this subroutine to deallocate the preconditioner data structure *prcs*, you must have called PSPGPR.
- You must deallocate *b*, *x*, sparse matrix *A*, and preconditioner data structure *prcs* before you deallocate the array descriptor *desc\_a*.
- PSPGPR allocates components of *prcs* as necessary. Prior to further calls to PSPGPR with the same *prcs* you must call PSPFREE; otherwise, there will be a memory leak.
- PSPALL allocates matrix *A* as necessary. Prior to further calls to PSPALL with the same matrix *A*, you must call PSPFREE; otherwise, there will be a memory leak.

## Error Conditions

### Computational Errors

None

### Resource Errors

None.

## Input-Argument and Miscellaneous Errors

### Stage 1:

1. *desc\_a* has not been initialized.

### Stage 2:

1. The BLACS context is invalid.

### Stage 3:

1. This subroutine was called from outside the process grid.

### Stage 4:

1. The process grid is not  $np \times 1$ .
2. The preconditioner data structure *prcs* is not valid.
3. The pointer components of *A* or *prcs* are not associated and therefore cannot be deallocated.

# PADFREE — Deallocates Space for an Array Descriptor for a General Sparse Matrix

## Purpose

This sparse utility subroutine deallocates space that is used for the array descriptor for a global general sparse matrix *A*.

## Syntax

Fortran	CALL PADFREE ( <i>desc_a</i> )
---------	--------------------------------

## On Entry

*desc\_a* is the array descriptor for the sparse matrix *A*.

Type: **required**

Specified as: the derived data type DESC\_TYPE.

## Notes and Coding Rules

1. Before you call this subroutine, you must have called PADALL.
2. You must deallocate *b*, *x*, sparse matrix *A*, and preconditioner data structure *prcs* before you deallocate the array descriptor *desc\_a*.
3. PADALL allocates *desc\_a* as necessary. Prior to further calls to PADALL with the same *desc\_a*, you must call PADFREE; otherwise, there will be a memory leak.

## Error Conditions

### Computational Errors

None

### Resource Errors

None.

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. *desc\_a* has not been initialized.

#### Stage 2:

1. The BLACS context is invalid.

#### Stage 3:

1. This subroutine was called from outside the process grid.

#### Stage 4:

1. The process grid is not  $np \times 1$ .

## Example—Using the Fortran 90 Sparse Subroutines

This example finds the solution to the linear system  $Ax = b$ . It also contains an application program that shows how you can use the Fortran 90 sparse linear algebraic equation subroutines and their utilities to solve this example.

The following is the general sparse matrix  $A$ :

$$\begin{bmatrix} 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & -1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 2.0 & -1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 2.0 & -1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 2.0 & -1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 2.0 & -1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 2.0 & -1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 2.0 \end{bmatrix}$$

The following is the dense vector  $b$ , containing the right-hand side:

$$\begin{bmatrix} 2.0 \\ 1.0 \\ 3.0 \\ 2.0 \\ 2.0 \\ 2.0 \\ 2.0 \\ 2.0 \\ 3.0 \end{bmatrix}$$

The following is the dense vector  $x$ , containing the initial guess to the solution:

$$\begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \end{bmatrix}$$

### Output

Global vector  $x$ :

$$\begin{array}{rcl} \text{B,D} & & 0 \\ & & \begin{bmatrix} 1.0 \\ 1.0 \\ 1.0 \\ --- \\ 1.0 \\ 1.0 \\ 1.0 \\ --- \\ 1.0 \\ 1.0 \\ 1.0 \end{bmatrix} \\ 0 & & \\ & & \\ 1 & & \\ & & \\ 2 & & \end{array}$$

The following is the  $3 \times 1$  process grid:

B,D	0
0	P <sub>00</sub>
1	P <sub>10</sub>
2	P <sub>20</sub>

Local vector  $x$ :

p,q	0
0	1.0 1.0 1.0
1	1.0 1.0 1.0
2	1.0 1.0 1.0

ITER = 4

ERR = 0.4071D-15

The value of *info* is 0 on all processes.

## Application Program

This application program illustrates how to use the Fortran 90 sparse linear algebraic equation subroutines and their utilities.

```
@process init(f90ptr)
!
! This program illustrates how to use the PESSL F90 Sparse Iterative
! Solver and its supporting utility subroutines. A very simple problem
! (DSRIS Example 1 from the ESSL Guide and Reference) using an
! HPF BLOCK data distribution is solved.
!
PROGRAM EXAMPLE90

! Interface module required to use the PESSL F90 Sparse Iterative Solver

USE F90SPARSE
IMPLICIT NONE

! Interface definition for the PARTS subroutine PART_BLOCK

INTERFACE PART_BLOCK
SUBROUTINE PART_BLOCK(GLOBAL_INDX,N,NP,PV,NV)
IMPLICIT NONE
INTEGER, INTENT(IN) :: GLOBAL_INDX, N, NP
INTEGER, INTENT(OUT) :: NV
INTEGER, INTENT(OUT) :: PV(*)
END SUBROUTINE PART_BLOCK
END INTERFACE

! Parameters
CHARACTER, PARAMETER :: ORDER='R'
INTEGER, PARAMETER :: IZERO=0, IONE=1
```

## Fortran 90 Example

```

! Sparse Matrices
  TYPE(D_SPMAT)                :: A, BLCK
! Preconditioner Data Structure
  TYPE(D_PRECN)                :: PRC

! Dense Vectors
  REAL(KIND(1.D0)), POINTER    :: B(:), X(:)

! Communications data structure
  TYPE(DESC_TYPE)              :: DESC_A

! BLACS parameters
  INTEGER                      :: NPROW, NPCOL, ICTXT, IAM, NP, MYROW, MYCOL

! Solver parameters
  INTEGER                      :: ITER, ITMAX, IERR, ITRACE,
&                               IPREC, METHD, ISTOPC, IPARM(20)
  REAL(KIND(1.D0))             :: ERR, EPS, RPARM(20)

! Other variables
  CHARACTER*5                  :: AFMT, ATYPE
  INTEGER                      :: IRCODE, IRCODE1, IRCODE2, IRCODE3
  INTEGER                      :: I,J
  INTEGER                      :: N,NNZERO
  INTEGER, POINTER             :: PV(:)
  INTEGER                      :: LPROCS, NROW, NCOL
  INTEGER                      :: GLOBAL_INDX, NV_COUNT
  INTEGER                      :: GLOBAL_INDX_OWNER, NV
  INTEGER                      :: LOCAL_INDX

!
!   Global Problem
!   DSRIS Example 1 from the ESSL Guide and Reference
!
  REAL*8                      :: A_GLOBAL(22),B_GLOBAL(9),XINIT_GLOBAL(9)
  INTEGER                      :: JA(22),IA(10)
  DATA A_GLOBAL               /2.D0,2.D0,-1.D0,1.D0,2.D0,1.D0,2.D0,-1.D0,
$                               1.D0,2.D0,-1.D0,1.D0,2.D0,-1.D0,1.D0,2.D0,
$                               -1.D0,1.D0,2.D0,-1.D0,1.D0,2.D0/
  DATA JA                     /1,2,3,2,3,1,4,5,4,5,6,5,6,7,6,7,8,
$                               7,8,9,8,9/
  DATA IA                     /1,2,4,6,9,12,15,18,21,23/

  DATA B_GLOBAL               /2.D0,1.D0,3.D0,2.D0,2.D0,2.D0,2.D0,2.D0,
$                               3.D0/
  DATA XINIT_GLOBAL           /0.D0,0.D0,0.D0,0.D0,0.D0,0.D0,0.D0,0.D0,
$                               0.D0/

! Initialize BLACS
! Define a NP x 1 Process Grid

  CALL BLACS_PINFO(IAM, NP)
  CALL BLACS_GET(IZERO, IZERO, ICTXT)
  CALL BLACS_GRIDINIT(ICTXT, ORDER, NP, IONE)
  CALL BLACS_GRIDINFO(ICTXT, NPROW, NPCOL, MYROW, MYCOL)

!
! Initialize the global problem size
!
  N = SIZE(IA)-1

!
! Guess for the local number of nonzeros
!
  NNZERO = SIZE(A_GLOBAL)

!
! Allocate and initialize some elements of the sparse matrix A

```



```

! its descriptor vector, DESC_A, the rhs vector B, and the
! solution vector X.
!
      CALL PADALL(N,PART_BLOCK,DESC_A,ICTXT)
      CALL PSPALL(A,DESC_A,NNZ=NNZERO)
      CALL PGEALL(B,DESC_A)
      CALL PGEALL(X,DESC_A)

!
! Allocate an integer work area to be used as an argument for
! the PART_BLOCK PARTS subroutine
!
      NROW = N
      NCOL = NROW
      LPROCS = MAX(NPROW, NROW + NCOL)
      ALLOCATE(PV(LPROCS), STAT = IRCODE)
      IF (IRCODE /= 0) THEN
        WRITE(6,*) 'PV Allocation failed'
        CALL BLACS_ABORT(ICTXT,-1)
        STOP
      ENDIF

! SETUP BLCK

      BLCK%M = 1
      BLCK%N = NCOL
      BLCK%FIDA = 'CSR'

      ALLOCATE(BLCK%AS(BLCK%N),STAT=IRCODE1)
      ALLOCATE(BLCK%IA1(BLCK%N),STAT=IRCODE2)
      ALLOCATE(BLCK%IA2(BLCK%M+1),STAT=IRCODE3)
      IRCODE = IRCODE1 + IRCODE2 + IRCODE3
      IF (IRCODE /= 0) THEN
        WRITE(6,*) 'Error allocating BLCK'
        CALL BLACS_ABORT(ICTXT,-1)
        STOP
      ENDIF

!
! In this simple example, all processes have a copy of
! the global sparse matrix, A, the global rhs vector, B,
! and the global initial guess vector, X.
!
! Each process will call PSPINS as many times as necessary
! to insert the local rows it owns.
!
! Each process will call PGEINS as many times as necessary
! to insert the local elements it owns.
!
      DO GLOBAL_INDX = 1, NROW
        CALL PART_BLOCK(GLOBAL_INDX,N,NP,PV,NV)
      DO
!
! In this simple example, NV will always be 1
! since there will not be duplicate coefficients
!
        DO NV_COUNT = 1, NV
          GLOBAL_INDX_OWNER = PV(NV_COUNT)
          IF (GLOBAL_INDX_OWNER == MYROW) THEN
            BLCK%IA2(1) = 1
            BLCK%IA2(2) = 1
            DO J = IA(GLOBAL_INDX), IA(GLOBAL_INDX+1)-1
              BLCK%AS(BLCK%IA2(2)) = A_GLOBAL(J)
              BLCK%IA1(BLCK%IA2(2)) = JA(J)
              BLCK%IA2(2) = BLCK%IA2(2) + 1
            ENDDO
            CALL PSPINS(A,GLOBAL_INDX,1,BLCK,DESC_A)
          ENDIF
        ENDDO
      ENDDO

```

## Fortran 90 Example

```
      CALL PGEINS(B,B_GLOBAL(GLOBAL_INDX:GLOBAL_INDX),
&          DESC_A,GLOBAL_INDX)
      CALL PGEINS(X,XINIT_GLOBAL(GLOBAL_INDX:GLOBAL_INDX),
&          DESC_A,GLOBAL_INDX)
&
&      ENDIF
&      END DO
&      END DO

! Assemble A and DESC_A
      AFMT = 'DEF'
      ATYPE = 'GEN'
      CALL PSPASB(A,DESC_A,MTYPE=ATYPE,
&          STOR=AFMT,DUPFLAG=2,INFO=IERR)

      IF (IERR /= 0) THEN
        IF (IAM.EQ.0) THEN
          WRITE(6,*) 'Error in assembly :',IERR
          CALL BLACS_ABORT(ICTXT,-1)
          STOP
        END IF
      END IF

! Assemble B and X

      CALL PGEASB(B,DESC_A)
      CALL PGEASB(X,DESC_A)

!
! Deallocate BLCK
!

      IF (ASSOCIATED(BLCK%AS))    DEALLOCATE(BLCK%AS)
      IF (ASSOCIATED(BLCK%IA1))   DEALLOCATE(BLCK%IA1)
      IF (ASSOCIATED(BLCK%IA2))   DEALLOCATE(BLCK%IA2)

!
! Deallocate Work vector
!
      IF (ASSOCIATED(PV))    DEALLOCATE(PV)

!
! Preconditioning
!
! We are using ILU for the preconditioner; PESSL
! will allocate PRC.
!

      IPREC = 2
      CALL PSPGPR(IPREC,A,PRC,DESC_A,INFO=IERR)

      IF (IERR /= 0) THEN
        IF (IAM.EQ.0) THEN
          WRITE(6,*) 'Error in preconditioner :',IERR
          CALL BLACS_ABORT(ICTXT,-1)
          STOP
        END IF
      END IF

!
! Iterative Solver - use the BICGSTAB method
!

      ITMAX = 1000
      EPS   = 1.D-8
      METHD = 1
      ISTOPC = 1
      ITRACE = 0
      IPARM   = 0
      IPARM(1) = METHD
      IPARM(2) = ISTOPC
```

```

IPARM(3) = ITMAX
IPARM(4) = ITRACE
RPARM    = 0.0D0
RPARM(1) = EPS

CALL PSPGIS(A,B,X,PRC,DESC_A,IPARM=IPARM,RPARM=RPARM,
&          INFO=IERR)

IF (IERR /= 0) THEN
  IF (IAM.EQ.0) THEN
    WRITE(6,*) 'Error in solver :',IERR
    CALL BLACS_ABORT(ICTXT,-1)
    STOP
  END IF
END IF

ITER = IPARM(5)
ERR  = RPARM(2)
IF (IAM.EQ.0) THEN
  WRITE(6,*) 'Number of iterations : ',ITER
  WRITE(6,*) 'Error on exit          : ',ERR
END IF

!
! Each process prints their local piece of the solution vector
!
  IF (IAM.EQ.0) THEN
    Write(6,*) 'Solution Vector X'
  END IF

  LOCAL_INDX = 1
  Do GLOBAL_INDX = 1, NROW
    CALL PART_BLOCK(GLOBAL_INDX,N,NP,PV,NV)
!
! In this simple example, NV will always be 1
! since there will not be duplicate coefficients
!
    DO NV_COUNT = 1, NV
      GLOBAL_INDX_OWNER = PV(NV_COUNT)
      IF (GLOBAL_INDX_OWNER == MYROW) THEN
        Write(6,*) GLOBAL_INDX, X(LOCAL_INDX)
        LOCAL_INDX = LOCAL_INDX + 1
      ENDIF
    END DO
  END DO

!
! Deallocate the vectors, the sparse matrix, and
! the preconditioner data structure.
! Finally, deallocate the descriptor vector
!
  CALL PGEFREE(B, DESC_A)
  CALL PGEFREE(X, DESC_A)
  CALL PSPFREE(A, DESC_A)
  CALL PSPFREE(PRC, DESC_A)
  CALL PADFREE(DESC_A)

!
! Terminate the process grid and the BLACS
!
  CALL BLACS_GRIDEXIT(ICTXT)
  CALL BLACS_EXIT(0)

END PROGRAM EXAMPLE90

```

---

## Fortran 77 Sparse Linear Algebraic Equation Subroutines and Their Utility Subroutines

This section contains the Fortran 77 sparse linear algebraic equation subroutine descriptions and their sparse utility subroutines.

## PADINIT — Initializes an Array Descriptor for a General Sparse Matrix

### Purpose

This sparse utility subroutine initializes an array descriptor, which is needed to establish a mapping between the global general sparse matrix  $A$  and its corresponding distributed memory location.

### Syntax

<b>Fortran</b>	CALL PADINIT ( <i>n</i> , <i>parts</i> , <i>desc_a</i> , <i>icontxt</i> )
<b>C and C++</b>	padinit ( <i>n</i> , <i>parts</i> , <i>desc_a</i> , <i>icontxt</i> );

### On Entry

*n* is the order of the global general sparse matrix  $A$  and the size of the index space.

Scope: **global**

Specified as: a fullword integer, where:  $n > 0$ .

*parts* is a user-supplied subroutine that specifies a mapping between a global index for an element in the global general sparse matrix  $A$  and its corresponding storage location on one or more processes.

Sample *parts* subroutines for common types of data distributions are shown in “Sample PARTS Subroutine” on page 956.

For details about how you must define the PARTS subroutine, see “Programming Considerations for the Parts Subroutine (Fortran 90 and Fortran 77)” on page 62.

Scope: **global**

Specified as: *parts* must be declared as an external subroutine in your application program. It can be whatever name you choose.

*desc\_a*

is the array descriptor for the global general sparse matrix  $A$ . DESC\_A(11), which is the length of the array descriptor, DLEN, is the only element that you must specify. To determine a sufficient value, see “Array Descriptor” on page 61.

Specified as: an array of length DLEN, containing fullword integers.

*icontxt* is the BLACS context parameter.

Scope: **global**

Specified as: a fullword integer that was returned in a prior call to BLACS\_GRIDINIT or BLACS\_GRIDMAP.

### On Return

*desc\_a* is the array descriptor for the global general sparse matrix  $A$ . This subroutine initializes the remaining elements in the array descriptor *desc\_a*. The elements of *desc\_a* are updated with subsequent calls to PDSPINS and finalized with a call to PDSPASB.

Table 33 on page 62 describes some of the elements of the array descriptor that you may want to reference. Your application programs should not

modify the elements of the array descriptor directly. The elements should only be updated with calls to PDSPINS and PDSPASB.

Returned as: an array of length DLEN, containing fullword integers.

## Notes and Coding Rules

1. Before you call this subroutine, you must create a  $np \times 1$  process grid, where  $np$  is the number of processes.
2. N\_ROW is stable after you have placed a call to this subroutine. N\_COL is stable after you have placed a call to PDSPASB. For more details about N\_ROW, N\_COL, and other elements of *desc\_a*, see Table 33 on page 62.

## Error Conditions

### Computational Errors

None.

### Resource Errors

None.

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. The BLACS context is invalid.

#### Stage 2:

1. This subroutine was called from outside the process grid.

#### Stage 3:

1. The process grid is not  $np \times 1$ .
2.  $n \leq 0$

#### Stage 4:

1. Each of the following global input arguments are checked to determine whether its value differs from the value specified on process P<sub>00</sub>:
  - $n$  differs.

#### Stage 5:

1.  $pv$  or  $nv$ , output from the user-supplied *parts* subroutine, was not valid. For valid values, see the appropriate argument description in “Programming Considerations for the Parts Subroutine (Fortran 90 and Fortran 77)” on page 62.
2. DLEN is too small. For valid values, see “Array Descriptor” on page 61.

## PDSPINIT — Initializes a General Sparse Matrix

### Purpose

This sparse utility subroutine initializes the local part of a general sparse matrix *A*.

### Syntax

<b>Fortran</b>	CALL PDSPINIT ( <i>as</i> , <i>ia1</i> , <i>ia2</i> , <i>infoa</i> , <i>desc_a</i> )
<b>C and C++</b>	pdspinit ( <i>as</i> , <i>ia1</i> , <i>ia2</i> , <i>infoa</i> , <i>desc_a</i> );

### On Entry

*as* See On Return.

*ia1* See On Return.

*ia2* See On Return.

*infoa* is an array, referred to as INFOA, providing more information about the general sparse matrix *A*. You must specify INFOA(1) through INFOA(3), as follows:

- INFOA(1) is the length of the array AS, where:  $\text{INFOA}(1) \geq \max(2, nnze)$ .
- INFOA(2) is the length of the array IA1, where:  
 $\text{INFOA}(2) \geq \max(3, (nnze + N\_ROW))$ .
- INFOA(3) is the length of the array IA2, where:  
 $\text{INFOA}(3) \geq \max(3, (nnze + N\_COL))$ .
- INFOA(4) through INFOA(30) are reserved for internal use.

*nnze* is the number of non-zero elements (including duplicate coefficients) in the local part of the global general sparse matrix *A*.

Specified as: an array of length 30, containing fullword integers.

*desc\_a* is the array descriptor for a global general sparse matrix *A* that is produced on a preceding call to PADINIT.

Specified as: an array of length DLEN, containing fullword integers.

### On Return

*as* is the local part, containing some internal values that are initialized by Parallel ESSL, of the global general sparse matrix *A*.

Scope: **local**

Returned as: a one-dimensional array of (at least) length INFOA(1), containing long-precision real numbers.

*ia1* is the local part, containing some internal values that are initialized by Parallel ESSL, of the sparse matrix indices.

Scope: **local**

Returned as: a one-dimensional array of (at least) length INFOA(2), containing fullword integers.

*ia2* is the local part, containing some internal values that are initialized by Parallel ESSL, of the sparse matrix indices.

Scope: **local**

Returned as: a one-dimensional array of (at least) length INFOA(3), containing fullword integers.

*infoa* is the array INFOA updated with some internal values that are set by Parallel ESSL.

Returned as: an array of length 30, containing fullword integers.

*desc\_a* is the updated array descriptor for the global general sparse matrix *A*.

Returned as: an array of length DLEN, containing fullword integers.

## Notes and Coding Rules

1. Before you call this subroutine, you must have called PADINIT.
2. For more details about N\_ROW, N\_COL, and other elements of *desc\_a*, see Table 33 on page 62.
3. For details about some of the elements stored in *infoa*, see Table 32 on page 60.

## Error Conditions

### Computational Errors

None

### Resource Errors

None.

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. The BLACS context is invalid.

#### Stage 2:

1. This subroutine was called from outside the process grid.

#### Stage 3:

1. The process grid is not  $np \times 1$ .
2. INFOA(1) < 2; that is, the size of AS < 2
3. INFOA(2) < 3; that is, the size of IA1 < 3
4. INFOA(3) < 3; that is, the size of IA2 < 3



## PDSPINS — Inserts Local Data into a General Sparse Matrix

### Purpose

This sparse utility subroutine is used by each process to insert all blocks of data it owns into its local part of the general sparse matrix *A*.

### Syntax

<b>Fortran</b>	CALL PDSPINS ( <i>as</i> , <i>ia1</i> , <i>ia2</i> , <i>infoa</i> , <i>desc_a</i> , <i>ia</i> , <i>ja</i> , <i>blcks</i> , <i>ib1</i> , <i>ib2</i> , <i>infob</i> )
<b>C and C++</b>	pdspins ( <i>as</i> , <i>ia1</i> , <i>ia2</i> , <i>infoa</i> , <i>desc_a</i> , <i>ia</i> , <i>ja</i> , <i>blcks</i> , <i>ib1</i> , <i>ib2</i> , <i>infob</i> );

### On Entry

*as* is the local part of the global general sparse matrix *A* that is produced on a preceding call to PDSPINIT or previous call(s) to this subroutine.

Scope: **local**

Specified as: a one-dimensional array of (at least) length INFOA(1), containing long-precision real numbers.

*ia1* is the local part of array IA1 that is produced on a preceding call to PDSPINIT or previous call(s) to this subroutine.

Scope: **local**

Specified as: a one-dimensional array of (at least) length INFOA(2), containing fullword integers.

*ia2* is the local part of the array IA2 that is produced on a preceding call to PDSPINIT or previous call(s) to this subroutine.

Scope: **local**

Specified as: a one-dimensional array of (at least) length INFOA(3), containing fullword integers.

*infoa* is the array INFOA that is produced on a preceding call to PDSPINIT or previous call(s) to this subroutine.

Specified as: an array of length 30, containing fullword integers.

*desc\_a* is the array descriptor for a global general sparse matrix *A* that is produced on a preceding call to PDSPINIT or previous call(s) to this subroutine.

Specified as: an array of length DLEN, containing fullword integers.

*ia* is the first global row index of the general sparse matrix *A* that receives data from the submatrix *BLCK*.

Scope: **local**

Specified as: a fullword integer;  $1 \leq ia \leq M$ .

*ja* is the first global column index of the general sparse matrix *A* that receives data from the submatrix *BLCK*.

Scope: **local**

Specified as: a fullword integer, where:  $ja = 1$ .

*blcks* is the local part of the sparse submatrix *BLCK*, referred to as BLCKS, to be inserted into the global general sparse matrix *A*. Each call to this

subroutine inserts one contiguous block of rows into the local part of the sparse matrix corresponding to the global submatrix  $A_{ia:ia+INFOB(6)-1, ja:ja+INFOB(7)-1}$ . This subroutine only can insert blocks of data it owns into its local part of the general sparse matrix  $A$ .

Scope: **local**

Specified as: a one-dimensional array of (at least) length  $INFOB(1)$ , containing long-precision real numbers.

*ib1* is an array, referred to as IB1, containing column numbers of each non-zero element in the submatrix **BLCK**.

Scope: **local**

Specified as: a one-dimensional array of (at least) length  $INFOB(2)$ , containing fullword integers.

*ib2* is the array, referred to as IB2, containing the starting positions of each row of the submatrix **BLCK** in array **BLCKS** and one position past the end of **BLCKS**.

Scope: **local**

Specified as: a one-dimensional array of (at least) length  $INFOB(3)$ , containing fullword integers:

- $IB2(1) = 1$
- $IB2(INFOB(6)+1) = 1+nz$  and  $nz$  is the **actual** number of non-zero elements in the submatrix **BLCK**.

*infob* is an array, referred to as INFOB, providing information about the submatrix **BLCK**. You must specify  $INFOB(1)$  through  $INFOB(7)$ , as follows:

- $INFOB(1)$  is the length of the array **BLCKS**, where:  $INFOB(1) \geq nz$ .
- $INFOB(2)$  is the length of the array **IB1**, where:  $INFOB(2) \geq nz$ .
- $INFOB(3)$  is the length of the array **IB2**, where:  $INFOB(3) \geq (INFOB(6)+1)$ .
- $INFOB(4)$  is the storage format of submatrix **BLCK**, where:  
If  $INFOB(4) = 1$ , submatrix **BLCK** is stored by rows.
- $INFOB(5)$  indicates the matrix type, where:  
If  $INFOB(5) = 1$ , **BLCK** is a general sparse matrix.
- $INFOB(6)$  is the number of local rows in the submatrix **BLCK**, where:  
 $1 \leq INFOB(6) \leq N\_ROW$ .
- $INFOB(7)$  is an upper bound on the number of local columns in the submatrix **BLCK**, where:  $1 \leq INFOB(7) \leq n$ .  $n$  is the order of the global general sparse matrix  $A$ .
- $INFOB(8)$  through  $INFOB(30)$  are reserved.

Specified as: an array of length 30, containing fullword integers.

## On Return

*as* is the updated local part of the global general sparse matrix  $A$ , updated with data from the submatrix **BLCK**.

Scope: **local**

Returned as: a one-dimensional array of (at least) length  $INFOA(1)$ , containing long-precision real numbers.

*ia1* is the updated local part of array **IA1**.

Scope: **local**

Returned as: a one-dimensional array of (at least) length `INFOA(2)`, containing fullword integers.

*ia2* is the updated local part of the array `IA2`.

Scope: **local**

Returned as: a one-dimensional array of (at least) length `INFOA(3)`, containing fullword integers.

*infoa* is the updated local part of array `INFOA`.

Returned as: an array of length 30, containing fullword integers.

*desc\_a* is the updated array descriptor for the global general sparse matrix `A`.

Returned as: an array of length `DLEN`, containing fullword integers.

## Notes and Coding Rules

1. Before you call this subroutine, you must have called `PADINIT` and `PDSPINIT`.
2. Arguments *BLCK* and *A* must not have common elements; otherwise, results are unpredictable.
3. For more details about `N_ROW`, `N_COL`, and other elements of *desc\_a*, see Table 33 on page 62.
4. For details about some of the elements stored in *infoa* or *infob*, see Table 32 on page 60.
5. The submatrix *BLCK* must be stored by rows; that is `INFOB(4) = 1`. For information about the storage-by-rows storage mode, see the *ESSL Guide and Reference*.
6. Each process has to call `PDSPINS` as many times as necessary to insert the local rows it owns. It is also possible to call `PDSPINS` multiple times to insert different or duplicate coefficients of the same local row it owns. For information on how duplicate coefficients are handled, see the *dupflag* argument description in `PDSPASB`. For an example of inserting coefficients of the same local row, see “Example” on page 650.

## Error Conditions

### Computational Errors

None.

### Resource Errors

None.

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. The BLACS context is invalid.

#### Stage 2:

1. This subroutine was called from outside the process grid.

#### Stage 3:

1. The process grid is not  $np \times 1$ .
2.  $ja \neq 1$
3. *desc\_a* is not valid.
4. The sparse matrix *A* is not valid.
5. `INFOB(4)  $\neq$  1`

6. INFOB(5)  $\neq$  1
7. INFOB(6) < 1 or INFOB(6) > N\_ROW
8. INFOB(7) < 1 or INFOB(7) > n
9. ia < 1 or ia > M
10. One or more rows to be inserted into submatrix A does not belong to the process.
11. DLEN is too small. For valid values, see “Array Descriptor” on page 61.
12. INFOB(1) < nz; that is, the size of BLCKS < nz
13. INFOB(2) < nz; that is, the size of IB1 < nz
14. INFOB(3) < (INFOB(6)+1); that is, the size of IB2 < (INFOB(6)+1)
15. INFOA(1) < max(2,nnze); that is, the size of AS < max(2,nnze)
16. INFOA(2) < max(3,(nnze+N\_ROW)); that is, the size of  
IA1 < max(3,(nnze+N\_ROW))
17. INFOA(3) < max(3,(nnze+N\_COL)); that is, the size of  
IA2 < max(3,(nnze+N\_COL))

## Examples

### Example

This piece of an example shows how to insert coefficients into the same GLOB\_ROW row by calling PDSPINS multiple times. This example would be useful in finite element applications, where PDSPINS inserts one element at a time into the global matrix, but more than one element may contribute to the same matrix row. In this case, PDSPINS is called with the same value of *ia* by all the elements contributing to that row.

For a complete example, see “Example—Using the Fortran 77 Sparse Subroutines” on page 669.

```

      .
      .
      .
DO GLOB_ROW = 1, N

      RINFOA(1) = 20
      RINFOA(2) = 20
      RINFOA(3) = 20
      RINFOA(4) = 1
      RINFOA(5) = 1
      RINFOA(6) = 1
      RINFOA(7) = N
      RIA2(1) = 1
      RIA2(2) = 2
      IA = GLOB_ROW

C      !      (x-1,y)
      RAS(1) = COEFF(X-1,Y,X,Y)
      RIA1(1) = IDX(X-1,Y)
      CALL PDSPINS(AS,IA1,IA2,INFOA,DESC_A,
+      IA,1,RAS,RIA1,RIA2,RINFOA)
C      !      (x,y-1)
      RAS(1) = COEFF(X,Y-1,X,Y)
      RIA1(1) = IDX(X,Y-1)
      CALL PDSPINS(AS,IA1,IA2,INFOA,DESC_A,
+      IA,1,RAS,RIA1,RIA2,RINFOA)
C      !      (x,y)
      RAS(1) = COEFF(X,Y,X,Y)
      RIA1(1) = IDX(X,Y)
      CALL PDSPINS(AS,IA1,IA2,INFOA,DESC_A,
+      IA,1,RAS,RIA1,RIA2,RINFOA)
C      !      (x,y+1)
```

```

      RAS(1) = COEFF(X,Y+1,X,Y)
      RIA1(1) = IDX(X,Y+1)
      CALL PDSPINS(AS,IA1,IA2,INFOA,DESC_A,
+      IA,1,RAS,RIA1,RIA2,RINFOA)

C      !      (x+1,y)
      RAS(1) = COEFF(X+1,Y,X,Y)
      RIA1(1) = IDX(X+1,Y)
      CALL PDSPINS(AS,IA1,IA2,INFOA,DESC_A,
+      IA,1,RAS,RIA1,RIA2,RINFOA)

END DO
      .
      .
      .

```

## PDGEINS — Inserts Local Data into a Dense Vector

### Purpose

This sparse utility subroutine is used by each process to insert all blocks of data it owns into its local part of the dense vector.

### Syntax

Fortran	CALL PDGEINS ( <i>nx</i> , <i>x</i> , <i>ldx</i> , <i>ix</i> , <i>jx</i> , <i>mb</i> , <i>nb</i> , <i>blcks</i> , <i>ldb</i> , <i>desc_a</i> )
C and C++	pdgeins ( <i>nx</i> , <i>x</i> , <i>ldx</i> , <i>ix</i> , <i>jx</i> , <i>mb</i> , <i>nb</i> , <i>blcks</i> , <i>ldb</i> , <i>desc_a</i> );

### On Entry

- nx* is the number of columns in the local dense vector.  
Scope: **local**  
Specified as: fullword integer;  $nx = 1$ .
- x* See On Return.
- ldx* is the local leading dimension of the local array.  
Scope: **local**  
Specified as: fullword integer;  $ldx \geq \max(1, N\_ROW)$ .
- ix* is the first global row index of the dense vector that receives data from the submatrix **BLCK**.  
Scope: **local**  
Specified as: a fullword integer;  $1 \leq ix \leq M$ .
- jx* is the first global column index of the dense vector that receives data from the submatrix **BLCK**.  
Scope: **local**  
Specified as: fullword integer;  $jx = 1$ .
- mb* is the number of local rows to be inserted into the dense vector.  
Scope: **local**  
Specified as: fullword integer;  $1 \leq mb \leq \min(N\_ROW, ldb)$ .
- nb* is the number of local columns to be inserted into the dense vector.  
Scope: **local**  
Specified as: fullword integer;  $nb = 1$ .
- blcks* is the local part, referred to as **BLCKS**, of the submatrix **BLCK**, containing the coefficients to be inserted into the dense vector. Each call to this subroutine inserts one contiguous block of data into the local part of the dense vector corresponding to the global submatrix  $X_{ix:ix+mb-1, jx:jx+nb-1}$ .  
Scope: **local**  
Specified as: an *ldb* by (at least) *nb* array, containing long-precision real numbers.
- ldb* is the local leading dimension for the local array **BLCKS**.

Scope: **local**

Specified as: fullword integer;  $ldb \geq \max(1, mb)$ .

*desc\_a* is the array descriptor that is produced on a preceding call to PADINIT, PDSPINIT, or PDSPINS.

Specified as: an array of length DLEN, containing fullword integers.

### On Return

*x* is the updated local part of the dense vector.

Scope: **local**

Returned as: an *ldx* by (at least) *nx* array, containing long-precision real numbers.

## Notes and Coding Rules

1. Before you call this subroutine, you must have called PADINIT.
2. You do not need a separate array descriptor for a dense vector because it must conform to the size of matrix *A*. For more details about N\_ROW, N\_COL, and other elements of *desc\_a*, see Table 33 on page 62.
3. This subroutine must be called for:
  - Vector *b* containing the right-hand side.
  - Vector *x* containing the initial guess to the solution.
4. Each process has to call PDGEINS as many times as necessary to insert the local elements it owns. It is also possible to call PDGEINS multiple times to insert different coefficients of the same local row it owns. Duplicate coefficients are overwritten.

## Error Conditions

### Computational Errors

None

### Resource Errors

1. None.

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. The BLACS context is invalid.

#### Stage 2:

1. This subroutine was called from outside the process grid.

#### Stage 3:

1. The process grid is not  $np \times 1$ .
2.  $nb \neq 1$
3.  $nx \neq 1$
4.  $jx \neq 1$
5. *desc\_a* is not valid.

#### Stage 4:

1.  $ldx < \max(1, N\_ROW)$
2.  $1 < mb$  or  $mb > N\_ROW$
3.  $ldb < \max(1, mb)$

## PDGEINS

4.  $ix < 1$  or  $ix > M$

### Stage 5:

1. One or more elements to be inserted into the submatrix *BLCK* does not belong to the process.



## PDSPASB — Assembles a General Sparse Matrix

### Purpose

This sparse utility subroutine uses the output from PDSPINS to assemble the global general sparse matrix  $A$  and its array descriptor *desc\_a*.

### Syntax

<b>Fortran</b>	CALL PDSPASB ( <i>as</i> , <i>ia1</i> , <i>ia2</i> , <i>infoa</i> , <i>desc_a</i> , <i>mtype</i> , <i>stor</i> , <i>dupflag</i> , <i>info</i> )
<b>C and C++</b>	pdspasb ( <i>as</i> , <i>ia1</i> , <i>ia2</i> , <i>infoa</i> , <i>desc_a</i> , <i>mtype</i> , <i>stor</i> , <i>dupflag</i> , <i>info</i> );

### On Entry

*as* is the local part of the global general sparse matrix  $A$  that is produced by previous call(s) to PDSPINS.

Scope: **local**

Specified as: a one-dimensional array of (at least) length INFOA(1), containing long-precision real numbers.

*ia1* is the local part of array IA1 that is produced by previous call(s) to PDSPINS.

Scope: **local**

Specified as: a one-dimensional array of (at least) length INFOA(2), containing fullword integers.

*ia2* is the local part of array IA2 that is produced by previous call(s) to PDSPINS.

Scope: **local**

Specified as: a one-dimensional array of (at least) length INFOA(3), containing fullword integers.

*infoa* is the array INFOA that is produced by previous call(s) to PDSPINS.

Specified as: an array of length 30, containing fullword integers.

*desc\_a* is the array descriptor for the global general sparse matrix  $A$  that is produced by previous call(s) to PDSPINS.

Specified as: an array of length DLEN, containing fullword integers.

*mtype* indicates the the form of the global sparse matrix  $A$  used, where:

If *mtype* = 'GEN',  $A$  is a general sparse matrix.

Scope: **global**

Specified as: a character variable of length 5; *mtype* = 'GEN'.

*stor* indicates the storage mode that the global general sparse matrix  $A$  is returned in, where:

If *stor* = 'DEF', this subroutine chooses an appropriate storage mode, which is an internal format accepted by the preconditioner and solver subroutines, for storing the global general sparse matrix  $A$  on output.

If *stor* = 'CSR', the global general sparse matrix  $A$  is stored in the storage-by-rows storage mode on output.

Scope: **global**

Specified as: a character variable of length 5; *stor* = 'DEF' or 'CSR'.

*dupflag*

is a flag indicating how to use coefficients that are specified more than once on the same process; that is, duplicate coefficients within the same local part of the matrix *A*:

If *dupflag* = 0, this subroutine uses the first of the duplicate coefficients.

If *dupflag* = 1, this subroutine adds all the duplicate coefficients with the same indices.

If *dupflag* = 2, this subroutine raises an error condition indicating that there are unexpected duplicate coefficients.

Scope: **global**

Specified as: a fullword integer; *dupflag* = 0, 1, or 2.

*info* See On Return.

### On Return

*as* is the updated local part of array AS of the global general sparse matrix *A*, where:

If *stor* = 'DEF', this subroutine chooses an appropriate storage mode, which is an internal format accepted by the preconditioner and solver subroutines, for storing the global general sparse matrix *A* on output.

If *stor* = 'CSR', the global general sparse matrix *A* is stored in the storage-by-rows storage mode on output.

Scope: **local**

Returned as: a one-dimensional array of (at least) length INFOA(1), containing long-precision real numbers.

*ia1* is the updated local part of array IA2.

Scope: **local**

Returned as: a one-dimensional array of (at least) length INFOA(2), containing fullword integers.

*ia2* is the updated local part of array IA2.

Scope: **local**

Returned as: a one-dimensional array of (at least) length INFOA(3), containing fullword integers.

*infoa* is the updated array INFOA.

Returned as: an array of length 30, containing fullword integers.

*desc\_a* is the final updated array descriptor for the global general sparse matrix *A*.

Returned as: an array of length DLEN, containing fullword integers.

*info* has the following meaning, when *info* is **present**:

If *info* = 0, then no input-argument errors or computational errors occurred. This indicates a normal exit.

**Note:** Because Parallel ESSL terminates the application if input-argument errors occur, the setting of *info* is irrelevant for these errors.

If  $info > 0$ , then one or more of the following computational errors occurred and the appropriate error messages were issued, indicating an error exit, where:

- If  $info = 1$ , the sparse matrix  $A$  contains duplicate coefficients.
- If  $info = 2$ , the sparse matrix  $A$  contains empty row(s).

Scope: **global**

Returned as: a fullword integer;  $info \geq 0$ .

## Notes and Coding Rules

1. In your C program,  $info$  must be passed by reference.
2. This subroutine accepts mixed case letters for the  $mttype$  and  $stor$  arguments.
3. Before you call this subroutine, you must have called PDSPINS as many times as needed; that is, you must have completed building the matrix with call(s) to PDSPINS before you place a call to this subroutine.
4. Your program must declare  $mttype$  and  $stor$  to be characters of length 5 with blanks padded to the right. C programs can use the fifth character for the null terminator.
5. For more details about  $N\_ROW$ ,  $N\_COL$ , and other elements of  $desc\_a$ , see Table 33 on page 62.
6. For details about some of the elements stored in  $infoa$ , see Table 32 on page 60.

## Error Conditions

### Computational Errors

The sparse matrix  $A$  contains duplicate coefficients or empty row(s). For details, see the description of the  $info$  argument.

### Resource Errors

1. Unable to allocate work space.
2. Unable to deallocate work space.

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. The BLACS context is invalid.

#### Stage 2:

1. This subroutine was called from outside the process grid.

#### Stage 3:

1. The process grid is not  $np \times 1$ .
2.  $desc\_a$  is not valid.
3. The sparse matrix  $A$  is not valid.
4.  $mttype \neq 'GEN'$
5.  $stor \neq 'DEF'$  or  $'CSR'$
6.  $dupflag \neq 0, 1$ , or  $2$
7. Some local rows in the sparse matrix  $A$  are missing.

#### Stage 4:

1.  $DLEN$  is too small. For valid values, see "Array Descriptor" on page 61.
2.  $INFOA(1) < \max(2, nnze)$ ; that is, the size of  $AS < \max(2, nnze)$

3.  $INFOA(2) < \max(3, (nnze + N\_ROW))$ ; that is, the size of  
 $IA1 < \max(3, (nnze + N\_ROW))$
4.  $INFOA(3) < \max(3, (nnze + N\_COL))$ ; that is, the size of  
 $IA2 < \max(3, (nnze + N\_COL))$

**Stage 5:**

1. Each of the following global input arguments are checked to determine whether its value differs from the value specified on process  $P_{00}$ :
  - *mtype* differs.
  - *stor* differs.
  - *dupflag* differs.

## PDGEASB — Assembles a Dense Vector

### Purpose

This sparse utility subroutine assembles a dense vector.

### Syntax

#### On Entry

- nx* is the number of columns in the local dense vector.  
 Scope: **local**  
 Specified as: fullword integer;  $nx = 1$ .
- x* is the local part of the dense matrix *x* produced by previous call(s) to PDGEINS.  
 Scope: **local**  
 Specified as: an *ldx* by (at least) *nx* array, containing long-precision real numbers.
- ldx* is the local leading dimension of the dense matrix.  
 Scope: **local**  
 Specified as: fullword integer;  $ldx \geq \max(1, N\_ROW)$ .
- desc\_a* is the array descriptor, which was finalized in a preceding call to PDSPASB.  
 Specified as: an array of length DLEN, containing fullword integers.

#### On Return

- x* is the updated local part of the dense matrix.  
 Scope: **local**  
 Returned as: an *ldx* by (at least) length *nx*, containing long-precision real numbers.

### Notes and Coding Rules

- Before you call this subroutine, you must have called PDGEINS as many times as needed; that is, you must have completed building the dense vectors with call(s) to PDGEINS before you place a call to this subroutine.  
 Before you call this subroutine, you must have called PDSPASB.
- You do not need a separate array descriptor for a dense vector because it must conform to the size of matrix *A*. For more details about *N\_ROW*, *N\_COL*, and other elements of *desc\_a*, see Table 33 on page 62.
- This subroutine must be called for:
  - Vector *b* containing the right-hand side.
  - Vector *x* containing the initial guess to the solution.

### Error Conditions

#### Computational Errors

None

## Resource Errors

None.

## Input-Argument and Miscellaneous Errors

### Stage 1:

1. The BLACS context is invalid.

### Stage 2:

1. This subroutine was called from outside the process grid.

### Stage 3:

1. The process grid is not  $np \times 1$ .
2. *desc\_a* is not valid.

### Stage 4:

1.  $ldx < \max(1, N\_ROW)$

## PDSPGPR — Preconditioner for a General Sparse Matrix

### Purpose

This subroutine computes a preconditioner for the global general sparse matrix  $A$  that should be passed unchanged to the PDSPGIS subroutine. The preconditioners include diagonal scaling or an incomplete LU factorization.

### Syntax

<b>Fortran</b>	CALL PDSPGPR ( <i>iprec</i> , <i>as</i> , <i>ia1</i> , <i>ia2</i> , <i>infoa</i> , <i>prcs</i> , <i>lprcs</i> , <i>desc_a</i> , <i>info</i> )
<b>C and C++</b>	pdspgpr ( <i>iprec</i> , <i>as</i> , <i>ia1</i> , <i>ia2</i> , <i>infoa</i> , <i>prcs</i> , <i>lprcs</i> , <i>desc_a</i> , <i>info</i> );

### On Entry

*iprec* is a flag that determines the type of preconditioning, where:

If *iprec* = 0, which is referred to as *none*, indicates the local part of the submatrix  $A$  is not preconditioned. PDSPGIS may not be effective in this case, unless the coefficient matrix is well conditioned; if your input matrix is not well conditioned, you should consider using *iprec* = 1 or 2.

If *iprec* = 1, which is referred to as *diagsc*, indicates the local part of the submatrix  $A$  is preconditioned by a local diagonal submatrix.

If *iprec* = 2, which is referred to as *ilu*, indicates the local part of the submatrix  $A$  is preconditioned by a local incomplete LU factorization.

It is suggested that you use a preconditioner. For an explanation, see “Notes and Coding Rules” on page 662.

Scope: **global**

Specified as: a fullword integer, where: *iprec* = 0, 1, or 2.

*as* is the local part of the global general sparse matrix  $A$ , finalized on a preceding call to PDSPASB.

Scope: **local**

Specified as: a one-dimensional array of (at least) length INFOA(1), containing long-precision real numbers.

*ia1* is the local part of array IA1 produced by a previous call to PDSPASB.

Scope: **local**

Specified as: a one-dimensional array of (at least) length INFOA(2), containing fullword integers.

*ia2* is the local part of array IA2 produced by a previous call to PDSPASB.

Scope: **local**

Specified as: a one-dimensional array of (at least) length INFOA(3), containing fullword integers.

*infoa* is the array INFOA produced by a previous call to PDSPASB.

Specified as: an array of length 30, containing fullword integers.

*prcs* See On Return.

*lprcs* is the length of array PRCS.

Scope: **local**

Specified as: fullword integer, where:

- If  $iprec = 0$ ,  $lprcs \geq 10$ .
- If  $iprec = 1$ ,  $lprcs \geq 10 + N\_ROW$ .
- If  $iprec = 2$ ,  $lprcs \geq 10 + 2(nnz) + N\_ROW + N\_COL + 40$

$nnz$  is the number of non-zero elements (without duplicate coefficients) in the local part of the global general sparse matrix  $A$ .

*desc\_a* is the array descriptor for the global general sparse matrix  $A$  that was finalized in a call to PDSPASB.

Specified as: an array of length DLEN, containing fullword integers.

*info* See On Return.

### On Return

*prcs* is the preconditioner data structure that must be pass unchanged to PDSPGIS.

Scope: **local**

Returned as: a one-dimensional array of (at least) length  $lprcs$ , containing long-precision real numbers.

*info* has the following meaning, when *info* is **present**:

If  $info = 0$ , then no input-argument errors or computational errors occurred. This indicates a normal exit.

**Note:** Because Parallel ESSL terminates the application if input-argument errors occur, the setting of *info* is irrelevant for these errors.

If  $info > 0$ , the value stored in *info* indicates the row index in the global general sparse matrix  $A$  where the preconditioner failed.

Scope: **global**

Returned as: a fullword integer;  $info \geq 0$ .

## Notes and Coding Rules

1. Before you call this subroutine, you must have called PDGEASB and PDSPASB.
2. In your C program, *info* must be passed by reference.
3. For more details about N\_ROW, N\_COL, and other elements of *desc\_a*, see Table 33 on page 62.
4. For details about some of the elements stored in *infoa* see Table 32 on page 60.
5. The convergence rate of an iterative method as applied to a given system of linear equations depends on the spectral properties of the coefficient matrix of the linear system; therefore it is often convenient to apply a linear transformation to the system such that the solution of the transformed system is the same (in exact arithmetic) as that of the original, but the spectral properties and the convergence behavior are more favorable. Such a transformation is called preconditioning. If a matrix  $M$  approximates  $A$ , then:
 
$$(M^{-1})Ax = (M^{-1})b$$

is a preconditioned system and  $M$  is called a preconditioner. In practice, the new coefficient matrix  $(M^{-1})A$  is almost never formed explicitly, but rather its action is computed during the application of the iterative method. The effectiveness of the preconditioning operation depends on a trade-off between



how well  $M$  approximates  $A$  and how costly it is to compute and invert it; no single preconditioner will give best overall performance under all situations. Note finally that it is quite rare for a linear system to behave well enough so as not to require preconditioning; indeed most linear systems originating from the discretization of difficult physical problems require preconditioning to have any convergence at all.

## Error Conditions

### Computational Errors

1. The preconditioner for the sparse matrix  $A$  is unstable. For details, see the *info* output argument for this subroutine.

### Resource Errors

1. Unable to allocate work space.

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. The BLACS context is invalid.

#### Stage 2:

1. This subroutine was called from outside the process grid.

#### Stage 3:

1. The process grid is not  $np \times 1$ .
2. *desc\_a* is not valid.
3. *iprec*  $\neq$  0, 1, or 2
4. *iprec* = 0 and *lprcs* < 10
5. *iprec* = 1 and *lprcs* < 10+N\_ROW
6. *iprec* = 2 and *lprcs* < 10+2(*nnz*)+N\_ROW+N\_COL+40
7. The storage format for  $A$  is not supported.

#### Stage 4:

1. Each of the following global input arguments are checked to determine whether its value differs from the value specified on process  $P_{00}$ :
  - *iprec* differs.

## PDSPGIS — Iterative Linear System Solver for a General Sparse Matrix

### Purpose

This subroutine solves a general sparse linear system of equations, using an iterative algorithm, with or without preconditioning. The methods include the more smoothly converging variant of the CGS method (Bi-CGSTAB), conjugate gradient squared (CGS), or transpose-free quasi-minimal residual method (TFQMR).

See references [7], [9], [12], and [36].

### Syntax

<b>Fortran</b>	CALL PDSPGIS ( <i>as</i> , <i>ia1</i> , <i>ia2</i> , <i>infoa</i> , <i>nrhs</i> , <i>b</i> , <i>ldb</i> , <i>x</i> , <i>ldx</i> , <i>prcs</i> , <i>desc_a</i> , <i>iparm</i> , <i>rparm</i> , <i>info</i> )
<b>C and C++</b>	pdspgis ( <i>as</i> , <i>ia1</i> , <i>ia2</i> , <i>infoa</i> , <i>nrhs</i> , <i>b</i> , <i>ldb</i> , <i>x</i> , <i>ldx</i> , <i>prcs</i> , <i>desc_a</i> , <i>iparm</i> , <i>rparm</i> , <i>info</i> );

### On Entry

*as* is the local part of the global general sparse matrix *A*, finalized on a preceding call to PDSPASB.

Scope: **local**

Specified as: a one-dimensional array of (at least) length INFOA(1), containing long-precision real numbers.

*ia1* is the local part of array IA1 produced by a previous call to PDSPASB.

Scope: **local**

Specified as: a one-dimensional array of (at least) length INFOA(2), containing fullword integers.

*ia2* is the local part of array IA2 produced by a previous call to PDSPASB.

Scope: **local**

Specified as: a one-dimensional array of (at least) length INFOA(3), containing fullword integers.

*infoa* is the array INFOA produced by a previous call to PDSPASB.

Specified as: an array of length 30, containing fullword integers.

*nrhs* the number of right-hand sides.

Scope: **global**

Specified as: a fullword integer; *nrhs* = 1.

*b* is the local part of the matrix *b*, containing the right-hand side of the matrix problem produced on a previous call to PDGEASB.

Scope: **local**

Specified as: an *ldb* by (at least) length *nrhs* array, containing long-precision real numbers.

*ldb* is the leading dimension of the local array B.

Scope: **local**

Specified as: a fullword integer; *ldb* ≥ max(1,N\_ROW)

- x* is the local part of the global vector *x*, containing the initial guess to the solution of the linear system and produced on a previous call to PDGEASB.
- Scope: **local**
- Specified as: an *ldx* by (at least) *nrhs* array, containing long-precision real numbers.
- ldx* is the leading dimension of the local array *x*.
- Scope: **local**
- Specified as: a fullword integer;  $ldx \geq \max(1, N\_ROW)$
- prcs* is the preconditioner data structure *prcs* produced by a previous call to PDSPGPR.
- Scope: **local**
- Specified as: a one-dimensional array of (at least) length *lprcs*, containing long-precision real numbers.
- desc\_a* is the array descriptor for the global general sparse matrix *A* that was finalized in a call to PDSPASB.
- Specified as: an array of length DLEN, containing fullword integers.
- iparm* is an array of parameters, IPARM(*i*), where:
- IPARM(1) is the flag, *methd*, used to select the iterative procedure used, where:
    - If IPARM(1) = 0, the following defaults are used:
      - *methd* = 1
      - *istopc* = 1
      - *itmax* = 500
      - *itrace* = 0
      - *eps* =  $10^{-8}$
    - If *methd* = 1, the more smoothly converging variant of the CGS method, referred to as Bi-CGSTAB, is used.
    - If *methd* = 2, the conjugate gradient squared method, referred to as CGS, is used.
    - If *methd* = 3, the transpose-free quasi-minimal residual method, referred to as TFQMR, is used.
  - IPARM(2) is the flag, *istopc* used to select the stopping criterion used in the computation, where the following items are used in the definitions of the stopping criteria below:
    - $\epsilon$  is the desired relative accuracy and is stored in *eps*
    - $x_j$  is the solution found at the *j*-th iteration.
    - $r_j$  and  $r_0$  are the preconditioned residuals obtained at iterations *j* and 0, respectively. (The residual at iteration *j* is defined as  $b - Ax_j$ .)
- If *istopc* = 1, the iterative method is stopped when:
- $\|r_j\|_2 / \|x_j\|_2 < \epsilon$
- If *istopc* = 2, the iterative method is stopped when:
- $\|r_j\|_2 / \|r_0\|_2 < \epsilon$
- If *istopc* = 3, the iterative method is stopped when:
- $\|x_j - x_{j-1}\|_2 / \|x_j\|_2 < \epsilon$

**Note:** Stopping criterion 3 performs poorly with the TFQMR method; therefore, if you specify TFQMR (*methd* = 3), you should not specify stopping criterion 3.

- IPARM(3) is the maximum number, *itmax*, of iterations allowed.
- IPARM(4), referred to as *itrace*, has the following meaning:  
If *itrace* = 0, then *itrace* is ignored.  
If *itrace* > 0, an informational message about convergence, which is based on the stopping criterion described in *istopc*, is issued at every *itrace*-th iteration and upon exit.
- IPARM(5), see On Return.
- IPARM(6) through IPARM(20) are reserved.

Scope: **global**

Specified as: an array of length 20, containing fullword integers, where:

- *methd* = 1, 2, or 3
- *istopc* = 1, 2, or 3.
- *itmax* ≥ 0.
- *itrace* ≥ 0.
- IPARM(6) through IPARM(20) should be set to zero.

*rparm*

is an array of parameters, RPARM(*i*), where:

- RPARM(1) is the relative accuracy  $\epsilon$ , referred to as *eps*, used in the stopping criterion.
- RPARM(2), see On Return.
- RPARM(3) through RPARM(20) are reserved.

Scope: **global**

Specified as: an array of length 20, containing long-precision real numbers, where:

- *eps* ≥ 0.
- RPARM(3) through RPARM(20) should be set to zero.

*info* See On Return.

## On Return

*x* is the local part of the solution vector *x*

Scope: **local**

Returned as: an array of (at least) length N\_ROW, containing long-precision real numbers.

*iparm* is an array of parameters, IPARM(*i*), where:

- IPARM(2) is the number of iterations, *iter*, performed by this subroutine.

Scope: **global**

Returned as: an array of length 20, containing fullword integers, where:

- *iter* ≥ 0

*rparm* is an array of parameters, RPARM(*i*), where:

- RPARM(2) contains the estimate of the error, *err*, of the solution, according to the stopping criterion, *istopc*, in use. For details, see the *istopc* argument description.

Scope: **global**

Returned as: an array of length 20, containing long-precision real numbers, where:

- $err \geq 0$

*info* has the following meaning, when *info* is **present**:

If  $info = 0$ , then no input-argument errors or computational errors occurred. This indicates a normal exit.

**Note:** Because Parallel ESSL terminates the application if input-argument errors occur, the setting of *info* is irrelevant for these errors.

If  $info > 0$ , then this subroutine exceeded *itmax* iterations without converging. You may want to try the following to get your matrix to converge:

1. You can increase the number of iterations and call this subroutine again without making any other changes to your program.
2. You can change the requested precision and/or the stopping criterion; your original precision requirement may be too stringent under a given stopping criterion.
3. You can use a preconditioner if you were not already doing so, or to change the one you were using. Note also that the efficiency of the preconditioner may depend on the data distribution strategy adopted. See “Notes and Coding Rules” on page 662.

Scope: **global**

Returned as: a fullword integer;  $info \geq 0$ .

## Notes and Coding Rules

1. In your C program, *info* must be passed by reference.
2. Before you call this subroutine, you must have called PDSPGPR.
3. For more details about N\_ROW, N\_COL, and other elements of *desc\_a*, see Table 33 on page 62.
4. For details about some of the elements stored in *infoa* see Table 32 on page 60.

## Error Conditions

### Computational Errors

1. This subroutine exceeded *itmax* iterations without converging. Vector *x* contains the approximate solution computed at the last iteration.

**Note:** If the preconditioner computed by PDSPGPR failed because the sparse matrix *A* is unstable, the results returned by this subroutine are unpredictable. For details, see the *info* output argument for PDSPGPR.

You may want to try the following to get your matrix to converge:

- a. You can increase the number of iterations and call this subroutine again without making any other changes to your program.
- b. You can change the requested precision and/or the stopping criterion; your original precision requirement may be too stringent under a given stopping criterion.

- c. You can use a preconditioner if you were not already doing so, or to change the one you were using. Note also that the efficiency of the preconditioner may depend on the data distribution strategy adopted. See “Notes and Coding Rules” on page 662.

### Resource Errors

1. Unable to allocate work space.

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. The BLACS context is invalid.

#### Stage 2:

1. This subroutine was called from outside the process grid.

#### Stage 3:

1. The process grid is not  $np \times 1$ .
2. *desc\_a* is not valid.
3. *nrhs*  $\neq 1$
4. *eps*  $< 0.0$
5. *methd*  $\neq 1, 2$ , or  $3$
6. The preconditioner data structure *prcs* is not valid.
7. *istopc*  $\neq 1, 2$ , or  $3$
8. *itmax*  $< 0$
9. *itrace*  $< 0$
10. The sparse matrix *A* is not valid.
11. The storage format for the sparse matrix *A* is not supported.

#### Stage 4:

1. *ldb*  $< \max(1, N\_ROW)$
2. *ldx*  $< \max(1, N\_ROW)$
3. The preconditioner data structure *prcs* is not valid.

#### Stage 5:

1. Each of the following global input arguments are checked to determine whether its value differs from the value specified on process  $P_{00}$ :
  - *iparm* differs.
  - *rparm* differs.
  - *eps* differs.
  - *methd* differs.
  - *istopc* differs.
  - *itmax* differs.
  - *itrace* differs.
  - Some element(s) of *prcs* differ.

## Example—Using the Fortran 77 Sparse Subroutines

This example finds the solution to the linear system  $Ax = b$ . It also contains an application program that shows how you can use the Fortran 77 sparse linear algebraic equation subroutines and their utilities to solve the problem shown in “Example—Using the Fortran 90 Sparse Subroutines” on page 636.

### Application Program

This application program illustrates how to use the Fortran 77 sparse linear algebraic equation subroutines and their utilities.

```
!
! This program illustrates how to use the PESSL F77 Sparse Iterative
! Solver and its supporting utility subroutines. A very simple problem
! (DSRIS Example 1 from the ESSL Guide and Reference) using an
! HPF BLOCK data distribution is solved.
!
PROGRAM EXAMPLE77

    IMPLICIT NONE

! Interface definition for the PARTS subroutine PART_BLOCK

    INTERFACE PART_BLOCK
        SUBROUTINE PART_BLOCK(GLOBAL_INDX,N,NP,PV,NV)
            IMPLICIT NONE
            INTEGER, INTENT(IN) :: GLOBAL_INDX, N, NP
            INTEGER, INTENT(OUT) :: NV
            INTEGER, INTENT(OUT) :: PV(*)
        END SUBROUTINE PART_BLOCK
    END INTERFACE

! External declaration for the PARTS subroutine PART_BLOCK
    EXTERNAL PART_BLOCK

! Parameters
    CHARACTER*1 ORDER
    CHARACTER*5 STOR
    CHARACTER*5 MTTYPE
    INTEGER*4 IZERO, IONE, DUPFLAG, N, NNZ
    PARAMETER (ORDER='R')
    PARAMETER (STOR='DEF')
    PARAMETER (MTTYPE='GEN')
    PARAMETER (IZERO=0)
    PARAMETER (IONE=1)
    PARAMETER (N=9)
    PARAMETER (NNZ=22)
    PARAMETER (DUPFLAG=2)

! Descriptor Vector
    INTEGER*4, ALLOCATABLE :: DESC_A(:)

! Sparse Matrices and related information
    REAL*8 AS(NNZ)
    INTEGER*4 IA1(NNZ+N), IA2(NNZ+N)
    INTEGER*4 INFOA(30)

    REAL*8 BS(NNZ)
    INTEGER*4 IB1(N+1), IB2(NNZ)
    INTEGER*4 INFOB(30)

! Preconditioner Data Structure
    REAL*8 PRCS(2*NNZ+2*N+41)

! Dense Vectors
```

## Fortran 77 Example

```

      REAL*8                B(N), X(N)

! BLACS parameters
      INTEGER*4            NPROW, NPCOL, ICTXT, IAM, NP, MYROW, MYCOL

! Solver parameters
      INTEGER*4            ITER, ITMAX, INFO, ITRACE,
&                          IPREC, METHD, ISTOPC, IPARM(20)
      REAL*8              ERR, EPS, RPARM(20)

! We will not have duplicates so PV used by the PARTS subroutine
! PART_BLOCK only needs to be of length 1.

      INTEGER              PV(1)

! Other variables
      INTEGER              IERR
      INTEGER              NB, LDB, LDBG
      INTEGER              NX, LDX, LDXG
      INTEGER              NRHS
      INTEGER              I,J
      INTEGER              GLOBAL_INDX, NV_COUNT
      INTEGER              GLOBAL_INDX_OWNER, NV
      INTEGER              LOCAL_INDX

!
!   Global Problem
!   DSRIS Example 1 from the ESSL Guide and Reference
!
      REAL*8              A_GLOBAL(NNZ), B_GLOBAL(N), XINIT_GLOBAL(N)
      INTEGER              JA(NNZ), IA(N+1)
      DATA A_GLOBAL      /2.D0,2.D0,-1.D0,1.D0,2.D0,1.D0,2.D0,-1.D0,
$                          1.D0,2.D0,-1.D0,1.D0,2.D0,-1.D0,1.D0,2.D0,
$                          -1.D0,1.D0,2.D0,-1.D0,1.D0,2.D0/
      DATA JA            /1,2,3,2,3,1,4,5,4,5,6,5,6,7,6,7,8,
$                          7,8,9,8,9/
      DATA IA            /1,2,4,6,9,12,15,18,21,23/

      DATA B_GLOBAL      /2.D0,1.D0,3.D0,2.D0,2.D0,2.D0,2.D0,2.D0,
$                          3.D0/
      DATA XINIT_GLOBAL  /0.D0,0.D0,0.D0,0.D0,0.D0,0.D0,0.D0,0.D0,
$                          0.D0/

! Initialize BLACS
! Define a NP x 1 Process Grid

      CALL BLACS_PINFO(IAM, NP)
      CALL BLACS_GET(IZERO, IZERO, ICTXT)
      CALL BLACS_GRIDINIT(ICTXT, ORDER, NP, IONE)
      CALL BLACS_GRIDINFO(ICTXT, NPROW, NPCOL, MYROW, MYCOL)

!
! Allocate the descriptor vector
!
      ALLOCATE(DESC_A(30 + 3*NP + 4*N + 3),STAT=IERR)
      IF (IERR.NE. 0) THEN
        WRITE(6,*) 'Error allocating DESC_A :',IERR
        CALL BLACS_ABORT(ICTXT,-1)
        STOP
      END IF

! Initialize some elements of the sparse matrix A
! and its descriptor vector, DESC_A
!

      DESC_A(11) = SIZE(DESC_A)
      CALL PADINIT(N,PART_BLOCK,DESC_A,ICTXT)

      INFOA(1) = SIZE(AS)

```



```

INFOA(2) = SIZE(IA1)
INFOA(3) = SIZE(IA2)
CALL PDSPINIT(AS,IA1,IA2,INFOA,DESC_A)

!
! In this simple example, all processes have a copy of
! the global sparse matrix, A, the global rhs vector B,
! and the global initial guess vector, X
!
! Each process will call PDSPINS as many times as necessary
! to insert the local rows it owns.
!
! Each process will call PDGEINS as many times as necessary
! to insert the local elements it owns.
!
      NB = 1
      LDB = SIZE(B,1)
      LDBG = SIZE(B_GLOBAL,1)
      NX = 1
      LDX = SIZE(X,1)
      LDXG = SIZE(XINIT_GLOBAL,1)

      DO GLOBAL_INDX = 1, N
        CALL PART_BLOCK(GLOBAL_INDX,N,NP,PV,NV)
      !
      ! In this simple example, NV will always be 1
      ! since there will not be duplicate coefficients
      !
      DO NV_COUNT = 1, NV
        GLOBAL_INDX_OWNER = PV(NV_COUNT)
        IF (GLOBAL_INDX_OWNER == MYROW) THEN
          IB2(1) = 1
          IB2(2) = 1
          DO J = IA(GLOBAL_INDX), IA(GLOBAL_INDX+1)-1
            BS(IB2(2)) = A_GLOBAL(J)
            IB1(IB2(2)) = JA(J)
            IB2(2) = IB2(2) + 1
          ENDDO
          INFOB(1) = IB2(2) - 1
          INFOB(2) = IB2(2) - 1
          INFOB(3) = 2
          INFOB(4) = 1
          INFOB(5) = 1
          INFOB(6) = 1
          INFOB(7) = N
          CALL PDSPINS(AS,IA1,IA2,INFOA,DESC_A,GLOBAL_INDX, 1,
&                     BS,IB1,IB2,INFOB)
          CALL PDGEINS(NB,B,LDB,GLOBAL_INDX,1,1,1,
&                     B_GLOBAL(GLOBAL_INDX),LDBG,DESC_A)
          CALL PDGEINS(NX,X,LDX,GLOBAL_INDX,1,1,1,
&                     XINIT_GLOBAL(GLOBAL_INDX),LDXG,DESC_A)
        ENDIF
      END DO
      END DO

! Assemble A and DESC_A
      CALL PDSPASB(AS,IA1,IA2,INFOA,DESC_A,
&                MTYPE,STOR,DUPFLAG,INFO)

      IF (INFO.NE. 0) THEN
        IF (IAM.EQ.0) THEN
          WRITE(6,*) 'Error in assembly :',INFO
          CALL BLACS_ABORT(1,CTXT,-1)
          STOP
        END IF
      END IF

```

## Fortran 77 Example

```
! Assemble B and X

      CALL PDGEASB(NB,B,LDB,DESC_A)
      CALL PDGEASB(NX,X,LDX,DESC_A)

!
! Preconditioning
!
! We are using ILU for the preconditioner
!
      IPREC = 2

      CALL PDSPGPR(IPREC,AS,IA1,IA2,INFOA,
&                PRCS,SIZE(P RCS),DESC_A,INFO)

      IF (INFO.NE. 0) THEN
        IF (IAM.EQ.0) THEN
          WRITE(6,*) 'Error in preconditioner :',INFO
          CALL BLACS_ABORT(ICTXT,-1)
          STOP
        END IF
      END IF

!
! Iterative Solver - use the BICGSTAB method
!
      NRHS = 1
      ITMAX = 1000
      EPS = 1.D-8
      METHD = 1
      ISTOPC = 1
      ITRACE = 0
      IPARM = 0
      IPARM(1) = METHD
      IPARM(2) = ISTOPC
      IPARM(3) = ITMAX
      IPARM(4) = ITRACE
      RPARM = 0.0D0
      RPARM(1) = EPS

      CALL PDSPGIS(AS,IA1,IA2,INFOA,NRHS,B,LDB,X,LDX,PRCS,DESC_A,
&                IPARM,RPARM,INFO)

      IF (INFO.NE. 0) THEN
        IF (IAM.EQ.0) THEN
          WRITE(6,*) 'Error in solver :',INFO
          CALL BLACS_ABORT(ICTXT,-1)
          STOP
        END IF
      END IF

      ERR = RPARM(2)
      ITER = IPARM(5)
      IF (IAM.EQ.0) THEN
        WRITE(6,*) 'Number of iterations : ',ITER
        WRITE(6,*) 'Error on exit : ',ERR
      END IF

!
! Each process prints their local piece of the solution vector
!
      IF (IAM.EQ.0) THEN
        Write(6,*) 'Solution Vector X'
      END IF

      LOCAL_INDX = 1
      Do GLOBAL_INDX = 1, N
```

```

      CALL PART_BLOCK(GLOBAL_INDX,N,NP,PV,NV)
!
! In this simple example, NV will always be 1
! since there will not be duplicate coefficients
!
      DO NV_COUNT = 1, NV
        GLOBAL_INDX_OWNER = PV(NV_COUNT)
        IF (GLOBAL_INDX_OWNER == MYROW) THEN
          Write(6,*) GLOBAL_INDX, X(LOCAL_INDX)
          LOCAL_INDX = LOCAL_INDX +1
        ENDIF
      END DO
END DO

!
! Deallocate the descriptor vector
!
      DEALLOCATE(DESC_A, STAT=IERR)
      IF (IERR .NE. 0) THEN
        WRITE(6,*) 'Error deallocating DESC_A :',IERR
        CALL BLACS_ABORT(ICTXT,-1)
        STOP
      END IF

!
! Terminate the process grid and the BLACS
!
      CALL BLACS_GRIDEXIT(ICTXT)
      CALL BLACS_EXIT(0)

      END PROGRAM EXAMPLE77

```



---

## Chapter 9. Eigensystem Analysis and Singular Value Analysis

This chapter describes the eigensystem analysis and singular value analysis subroutines.

---

### Overview of the Eigensystem Analysis and Singular Value Analysis Subroutines

The eigensystems analysis subroutines provide solutions to the algebraic and generalized eigensystem analysis problem. The singular value analysis subroutines provide the singular value decomposition. These subroutines include a subset of the ScaLAPACK subroutines. See references [20] and [21].

**Note:** These subroutines were designed in accordance with the proposed ScaLAPACK standard. If these subroutines do not comply with the standard as approved, IBM will consider updating them to do so. If IBM updates these subroutines, the update could require modifications of the calling application program.

*Table 115. List of Eigensystem Analysis and Singular Value Analysis Subroutines*

Descriptive Name	Long-Precision Subroutine	Page
Selected Eigenvalues and, Optionally, the Eigenvectors of a Real Symmetric or Complex Hermitian Matrix	PDSYEVX PZHHEEVX	677
Selected Eigenvalues and, Optionally, the Eigenvectors of a Real Symmetric or Complex Hermitian Positive Definite Generalized Eigenproblem	PDSYGVX PZHEGVX	698
Reduce a Real Symmetric or Complex Hermitian Matrix to Tridiagonal Form	PDSYTRD PZHETRD	724
Reduce a Real Symmetric or Complex Hermitian Positive Definite Generalized Eigenproblem to Standard Form	PDSYGST PZHEGST	739
Reduce a General Matrix to Upper Hessenberg Form	PDGEHRD	753
Reduce a General Matrix to Bidiagonal Form	PDGEBRD PZGEBRD	762
Singular Value Decomposition of a General Matrix	PDGESVD PZGESVD	780

---

## Eigensystem Analysis and Singular Value Analysis Subroutines

This section contains the eigensystem analysis subroutine descriptions.

## PDSYEVX and PZHEEVX — Selected Eigenvalues and, Optionally, the Eigenvectors of a Real Symmetric or Complex Hermitian Matrix

### Purpose

These subroutines compute selected eigenvalues and, optionally, the eigenvectors of a real symmetric or complex Hermitian matrix  $A$ , where  $A$  represents the global real symmetric or complex Hermitian submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$ . Eigenvalues and eigenvectors can be selected by specifying a range of values or a range of indices for the eigenvalues.

If  $n = 0$ , no computation is performed and the subroutine returns after doing some parameter checking.

See references [13], [25], [26], and [27].

Table 116. Data Types

<i>vl, vu, abstol, orfac, w, rwork, gap</i>	<i>A, Z, work</i>	<i>iwork, ifail, iclustr</i>	Subroutine
Long-precision real	Long-precision real	Integer	PDSYEVX
Long-precision real	Long-precision complex	Integer	PZHEEVX

### Syntax

Fortran	CALL PDSYEVX ( <i>jobz, range, uplo, n, a, ia, ja, desc_a, vl, vu, il, iu, abstol, m, nz, w, orfac, z, iz, jz, desc_z, work, lwork, iwork, liwork, ifail, iclustr, gap, info</i> )
	CALL PZHEEVX ( <i>jobz, range, uplo, n, a, ia, ja, desc_a, vl, vu, il, iu, abstol, m, nz, w, orfac, z, iz, jz, desc_z, work, lwork, rwork, lrwork, iwork, liwork, ifail, iclustr, gap, info</i> )
C and C++	pdsyevx ( <i>jobz, range, uplo, n, a, ia, ja, desc_a, vl, vu, il, iu, abstol, m, nz, w, orfac, z, iz, jz, desc_z, work, lwork, iwork, liwork, ifail, iclustr, gap, info</i> );
	pzheevx ( <i>jobz, range, uplo, n, a, ia, ja, desc_a, vl, vu, il, iu, abstol, m, nz, w, orfac, z, iz, jz, desc_z, work, lwork, rwork, lrwork, iwork, liwork, ifail, iclustr, gap, info</i> );

### On Entry

*jobz* indicates the type of computation to be performed, where:

If *jobz* = 'N', eigenvalues only are computed.

If *jobz* = 'V', eigenvalues and eigenvectors are computed.

Scope: **global**

Specified as: a single character; *jobz* = 'N' or 'V'.

*range* indicates which eigenvalues to compute, where:

If *range* = 'A', all eigenvalues are to be found.

If *range* = 'V', all eigenvalues in the interval [*vl*, *vu*] are to be found.

If *range* = 'I', the *il*-th through *iu*-th eigenvalues are to be found.

Scope: **global**

Specified as: a single character; *range* = 'A', 'V', or 'I'.

*uplo* indicates whether the upper or lower triangular part of the global submatrix  $A$  is referenced, where:

If *uplo* = 'U', the upper triangular part is referenced.

If *uplo* = 'L', the lower triangular part is referenced.

Scope: **global**

Specified as: a single character; *uplo* = 'U' or 'L'.

*n* is the order of submatrix *A* used in the computation.

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

*a* is the local part of the global real symmetric or complex Hermitian matrix *A*. This identifies the **first element** of the local array *A*. This subroutine computes the location of the first element of the local subarray used, based on *ia*, *ja*, *desc\_a*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading LOCp(*ia*+*n*-1) by LOCq(*ja*+*n*-1) part of the local array *A* must contain the local pieces of the leading *ia*+*n*-1 by *ja*+*n*-1 part of the global matrix, and:

- If *uplo* = 'U', the leading  $n \times n$  upper triangular part of the global submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  must contain the upper triangular part of the matrix, and the strictly lower triangular part is not referenced.
- If *uplo* = 'L', the leading  $n \times n$  lower triangular part of the global submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  must contain the lower triangular part of the matrix, and the strictly upper triangular part is not referenced.

Scope: **local**

Specified as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 116 on page 677. Details about the square block-cyclic data distribution of global matrix *A* are stored in *desc\_a*.

*ia* is the row index of the global matrix *A*, identifying the first row of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq M\_A$  and  $ia+n-1 \leq M\_A$ .

*ja* is the column index of the global matrix *A*, identifying the first column of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$  and  $ja+n-1 \leq N\_A$ .

*desc\_a* is the array descriptor for global matrix *A*, described in the following table:

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global



<i>desc_a</i>	Name	Description	Limits	Scope
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq \text{RSRC\_A} < p$	Global
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq \text{CSRC\_A} < q$	Global
9	LLD_A	The leading dimension of the local array	$\text{LLD\_A} \geq \max(1, \text{LOCp}(\text{M\_A}))$	Local

Specified as: an array of (at least) length 9, containing fullword integers.

*vl* has the following meaning:

If *range* = 'V', it is the lower bound of the interval to be searched for eigenvalues.

If *range*  $\neq$  'V', this argument is ignored.

Scope: **global**

Specified as: a number of the data type indicated in Table 116 on page 677.

If *range* = 'V',  $vl < vu$ .

*vu* has the following meaning:

If *range* = 'V', it is the upper bound of the interval to be searched for eigenvalues.

If *range*  $\neq$  'V', this argument is ignored.

Scope: **global**

Specified as: a number of the data type indicated in Table 116 on page 677.

If *range* = 'V',  $vl < vu$ .

*il* has the following meaning:

If *range* = 'T', it is the index (from smallest to largest) of the smallest eigenvalue to be returned.

If *range*  $\neq$  'T', this argument is ignored.

Scope: **global**

Specified as: a fullword integer;  $il \geq 1$ .

*iu* has the following meaning:

If *range* = 'T', it is the index (from smallest to largest) of the largest eigenvalue to be returned.

If *range*  $\neq$  'T', this argument is ignored.

Scope: **global**

Specified as: a fullword integer;  $\min(il, n) \leq iu \leq n$ .

*abstol* is the absolute tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval  $[a, b]$  of width less than or equal to:

- $abstol + \epsilon(\max(|a|, |b|))$

where  $\epsilon$  is the machine precision. If *abstol* is less than or equal to zero, then  $\epsilon(\text{norm}(T))$  is used in its place, where  $\text{norm}(T)$  is the 1-norm of the

tridiagonal matrix obtained by reducing  $A$  to tridiagonal form. For most problems, this is the appropriate level of accuracy to request.

For certain strongly graded matrices, greater accuracy can be obtained in very small eigenvalues by setting *abstol* to a very small positive number. However, if *abstol* is less than:

$$\sqrt{unfl}$$

where *unfl* is the underflow threshold, then:

$$\sqrt{unfl}$$

is used in its place.

Eigenvalues are computed most accurately when *abstol* is set to twice the underflow threshold—that is,  $(2)(unfl)$ .

If *jobz* = 'V', then setting *abstol* to *unfl*, the underflow threshold, yields the most orthogonal eigenvectors.

**Note:**

- $\epsilon$  is approximately 0.222044604925031308E-15
- *unfl* is approximately 0.222507385850720138E-307
- $\sqrt{unfl}$  is approximately 0.149166814624004135E-153

Scope: **global**

Specified as: a number of the data type indicated in Table 116 on page 677.

*m* See On Return.

*nz* See On Return.

*w* See On Return.

*orfac* specifies which eigenvectors should be reorthogonalized. Eigenvectors that correspond to eigenvalues which are within:

- $ortol = (orfac)(\text{norm}(A))$

of each other (where  $\text{norm}(A)$  is the 1-norm of  $A$ ) are to be reorthogonalized.

However, if the workspace is insufficient (see *lwork* and *lrwork*), *ortol* may be decreased until all eigenvectors to be reorthogonalized can be stored in one process.

If *orfac* is zero, no reorthogonalization is done.

If *orfac* is less than zero, a default value of  $10^{-3}$  is used.

Scope: **global**

Specified as: a number of the data type indicated in Table 116 on page 677.

*z* See On Return.

*iz* is the row index of the global matrix *Z*, identifying the first row of the submatrix *Z*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq iz \leq M\_Z$  and  $iz+n-1 \leq M\_Z$ .

*jz* is the column index of the global matrix *Z*, identifying the first column of the submatrix *Z*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq jz \leq N\_Z$  and  $jz+n-1 \leq N\_Z$ .

*desc\_z* is the array descriptor for global matrix *Z*, described in the following table:

<i>desc_z</i>	Name	Description	Limits	Scope
1	DTYPE_Z	Descriptor type	DTYPE_Z=1	Global
2	CTXT_Z	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_Z	Number of rows in the global matrix	If $n = 0$ : $M\_Z \geq 0$ Otherwise: $M\_Z \geq 1$	Global
4	N_Z	Number of columns in the global matrix	If $n = 0$ : $N\_Z \geq 0$ Otherwise: $N\_Z \geq 1$	Global
5	MB_Z	Row block size	$MB\_Z \geq 1$	Global
6	NB_Z	Column block size	$NB\_Z \geq 1$	Global
7	RSRC_Z	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_Z < p$	Global
8	CSRC_Z	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_Z < q$	Global
9	LLD_Z	The leading dimension of the local array	$LLD\_Z \geq \max(1, LOCp(M\_Z))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

*work* has the following meaning:

If  $lwork = 0$ , *work* is ignored.

If  $lwork \neq 0$ , *work* is a work area used by this subroutine, where:

- If  $lwork \neq -1$ , then its size is (at least) of length *lwork*.
- If  $lwork = -1$ , then its size is (at least) of length 1.

Scope: **local**

Specified as: an area of storage containing numbers of data type indicated in Table 116 on page 677.

*lwork* is the number of elements in array WORK.

Scope:

- If  $lwork \geq 0$ ,  $lwork$  is **local**.
- If  $lwork = -1$ ,  $lwork$  is **global**.

Specified as: a fullword integer; where:

- If  $lwork = 0$ , PDSYEVX and PZHEEVX dynamically allocate the work area used by the subroutine. The work area is deallocated before control is returned to the calling program. This option is an extension to the ScaLAPACK standard.
- If  $lwork = -1$ , PDSYEVX and PZHEEVX perform a work area query and return the minimum required size of  $work$  in  $work_1$ . No computation is performed and the subroutine returns after error checking is complete.
- Otherwise, use the following rules to determine the value to specify:
  - If  $jobz = 'N'$ , it must have the following value:  
 For PDSYEVX,  $lwork \geq 3(n+ja-1)+2n + \max(5nn, (nb)(np+1))$   
 For PZHEEVX,  $lwork \geq n+ja-1 + \max(3nb, nb(np+1))$
  - If  $jobz = 'V'$ , then the amount of workspace required to guarantee that all eigenvectors are computed is:  
 For PDSYEVX,  $lwork \geq 3(n+ja-1)+2n + \max(5nn, (np0)(mq0)+2(nb)(nb)) + \text{iceil}(neig, (nprow)(npcol))(nn)$   
 For PZHEEVX,  $lwork \geq n+ja-1 + (np0+mq0+nb)(nb)$

where:

- $nn = \max(n, nb, 2)$
- $neig$  is the number of eigenvectors requested
- $nb = MB\_A = NB\_A = MB\_Z = NB\_Z$
- $np = \text{NUMROC}(nn, nb, myrow, iarow, nprow)$
- $np0 = \text{NUMROC}(nn+iroffz, nb, izrow, izrow, nprow)$
- $mq0 = \text{NUMROC}(\max(neig, nb, 2)+icoffz, nb, izcol, izcol, npcol)$
- $iarow = \text{mod}(\text{RSRC\_A} + (ia-1)/nb, nprow)$
- $izrow = \text{mod}(\text{RSRC\_Z} + (iz-1)/nb, nprow)$
- $izcol = \text{mod}(\text{CSRC\_Z} + (jz-1)/nb, npcol)$
- $iroffz = \text{mod}(iz-1, MB\_Z)$
- $icoffz = \text{mod}(jz-1, NB\_Z)$

For PDSYEVX, the computed eigenvectors may not be orthogonal if the minimum workspace is supplied and  $ortol$  is too small; therefore, if you want to guarantee orthogonality (at the cost of potentially compromising performance), you should add the following to  $lwork$ :

$(clustersize-1)(n)$

where  $clustersize$  is the number of eigenvalues in the largest cluster, where a cluster is defined as a set of close eigenvalues:

- $\{w_k, \dots, w_{k+clustersz-1} \mid w_{j+1} \leq w_j + orfac(2)(\text{norm}(A))\}$

**Note:** PDSYEVX does **not** add this amount when dynamically allocating this workspace. You must use static allocation if you want to guarantee orthogonality.

When  $lwork$  is too small:

- For PDSYEVX, if  $lwork$  is too small to guarantee orthogonality, this subroutine attempts to maintain orthogonality in the clusters with the smallest spacing between the eigenvalues.
- If  $lwork$  is too small to compute all the eigenvectors requested, no computation is performed, except that if  $range = 'V'$ , this subroutine does not know how many eigenvectors are requested until the eigenvalues are computed. Therefore, if  $range = 'V'$  and  $lwork$  is large

enough to allow this subroutine to compute the eigenvalues, this subroutine computes the eigenvalues and as many eigenvectors as it can.

For the relationship between workspace, orthogonality, and performance, see Notes and Coding Rules 14 on page 689.

*rwork* has the following meaning:

If *lrwork* = 0, *rwork* is ignored.

If *lrwork* ≠ 0, *rwork* is a work area used by this subroutine, where:

- If *lrwork* ≠ -1, then its size is (at least) of length *lrwork*.
- If *lrwork* = -1, then its size is (at least) of length 1.

Scope: **local**

Specified as: an area of storage containing numbers of data type indicated in Table 116 on page 677.

*lrwork* is the number of elements in array RWORK.

Scope:

- If *lrwork* ≥ 0, *lrwork* is **local**.
- If *lrwork* = -1, *lrwork* is **global**.

Specified as: a fullword integer; where:

- If *lrwork* = 0, PZHEEVX dynamically allocates the work area used by this subroutine. The work area is deallocated before control is returned to the calling program. This option is an extension to the ScaLAPACK standard.
- If *lrwork* = -1, PZHEEVX performs a work area query and return the minimum required size of *rwork* in *rwork*<sub>1</sub>. No computation is performed and the subroutine returns after error checking is complete.
- Otherwise, use the following rules to determine the value to specify:
  - If *jobz* = 'N', it must have the following value:
 
$$lrwork \geq 2(n+ja-1) + 2n + 5nn$$
  - If *jobz* = 'V', then the amount of workspace required to guarantee that all eigenvectors are computed is:
 
$$lrwork \geq 2(n+ja-1) + 2n + \max(5nn, (np0)(mq0)) + \text{iceil}(neig, (nprow)(npcol))(nn)$$

where:

- $nn = \max(n, nb, 2)$
- *neig* is the number of eigenvectors requested
- $nb = MB\_A = NB\_A = MB\_Z = NB\_Z$
- $np0 = \text{NUMROC}(nn + iroffz, nb, izrow, izrow, nprow)$
- $mq0 = \text{NUMROC}(\max(neig, nb, 2) + icoffz, nb, izcol, izcol, npcol)$
- $izrow = \text{mod}(\text{RSRC\_Z} + (iz-1)/nb, nprow)$
- $izcol = \text{mod}(\text{CSRC\_Z} + (jz-1)/nb, npcol)$
- $iroffz = \text{mod}(iz-1, MB\_Z)$
- $icoffz = \text{mod}(jz-1, NB\_Z)$

For PZHEEVX, the computed eigenvectors may not be orthogonal if the minimum workspace is supplied and *ortol* is too small; therefore, if you want to guarantee orthogonality (at the cost of potentially compromising performance), you should add the following to *lrwork*:

$(clustersize-1)(n)$

where *clustersize* is the number of eigenvalues in the largest cluster, where a cluster is defined as a set of close eigenvalues:

$$- \{w_k, \dots, w_{k+clustersz-1} \mid w_{j+1} \leq w_j + orfac(2)(\text{norm}(A))\}$$

**Note:** PZHEEVX does **not** add this amount when dynamically allocating this workspace. You must use static allocation if you want to guarantee orthogonality.

When *lwork* is too small:

- If *lwork* is too small to guarantee orthogonality, this subroutine attempts to maintain orthogonality in the clusters with the smallest spacing between the eigenvalues.
- If *lwork* is too small to compute all the eigenvectors requested, no computation is performed, except that if *range* = 'V', this subroutine does not know how many eigenvectors are requested until the eigenvalues are computed. Therefore, if *range* = 'V' and *lwork* is large enough to allow this subroutine to compute the eigenvalues, this subroutine computes the eigenvalues and as many eigenvectors as it can.

For the relationship between workspace, orthogonality, and performance, see Notes and Coding Rules 14 on page 689.

*iwork* has the following meaning:

If *liwork* = 0, *iwork* is ignored.

If *liwork* ≠ 0, *iwork* is a work area used by this subroutine, where:

- If *liwork* ≠ -1, then its size is (at least) of length *liwork*.
- If *liwork* = -1, then its size is (at least) of length 1.

Scope: **local**

Specified as: an area of storage containing fullword integers.

*liwork* is the number of elements in array IWORK.

Scope:

- If *liwork* ≥ 0, *liwork* is **local**.
- If *liwork* = -1, *liwork* is **global**.

Specified as: a fullword integer; where:

- If *liwork* = 0, PDSYEVX and PZHEEVX dynamically allocate the work area used by the subroutine. The work area is deallocated before control is returned to the calling program. This option is an extension to the ScaLAPACK standard.
- If *liwork* = -1, PDSYEVX and PZHEEVX perform a work area query and return the minimum required size of *iwork* in *iwork*<sub>1</sub>. No computation is performed and the subroutine returns after error checking is complete.
- Otherwise, it must have the following value:

$$liwork \geq \max(isizestein, isizestebz) + 2n$$

where:

- *isizestein* must have the following value:
  - If *jobz* = 'N', *isizestein* = 3n+nprocs+1
  - If *jobz* = 'V', *isizestein* = (n+jz-1) + 2n+nprocs+1
- *isizestebz* = max(4n, 14, nprocs)
- *nprocs* = (nprow)(npcol)

*ifail* See On Return.

*iclustr* See On Return.

*gap* See On Return.

*info* See On Return.

## On Return

*a* *a* is the local part of the global matrix *A*, where:

If *uplo* = 'U', the upper triangle and diagonal of submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  are overwritten; that is, the original input is not preserved.

If *uplo* = 'L', the lower triangle and diagonal of submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  are overwritten; that is, the original input is not preserved.

Scope: **local**

Returned as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 116 on page 677.

*m* is the number of eigenvalues found.

Scope: **global**

Returned as: a fullword integer;  $0 \leq m \leq n$ .

*nz* has the following meaning:

If *jobz*  $\neq$  'V', then *nz* is ignored.

If *jobz* = 'V', then *nz* is the number of eigenvectors computed—that is, the number of columns of *Z* used in the computation. On output, *nz* = *m* unless you provide insufficient space. To get all the eigenvectors requested, you must supply both sufficient space to hold the eigenvectors in *Z* and sufficient workspace to compute them (see *lwork* and *lrwork*).

If *range* = 'A' or 'T', this subroutine does not perform any computations if the work space supplied is insufficient. In this case, an input-argument error is issued and your job is terminated. For *range* = 'V', the number of requested eigenvectors is unknown until the eigenvalues are found. In this case, the subroutine computes as many eigenvectors as space allows. Then, if *nz*  $\neq$  *m*, a computational error message is issued.

Scope: **global**

Returned as: a fullword integer;  $0 \leq nz \leq m$ .

*w* On normal exit (see *info*), it is the vector *w*, containing the selected eigenvalues in ascending order in the first *m* elements of *w*.

Scope: **global**

Returned as: a one-dimensional array of (at least) length *n*, containing numbers of the data type indicated in Table 116 on page 677.

*z* has the following meaning:

If *jobz* = 'N', then *z* is ignored.

If *jobz* = 'V' and there is a normal exit (see *info*), then this is the updated local part of the global matrix *Z*, where columns *jz* to *jz+m-1* of the global matrix *Z* contain the orthonormal eigenvectors of the global matrix *A*, corresponding to the selected eigenvalues. If an eigenvector fails to converge, then the corresponding column of the global matrix *Z* contains the last approximation to the eigenvector, and the index of the eigenvector is returned in *ifail*.

This identifies the **first element** of the local array Z. This subroutine computes the location of the first element of the local subarray used, based on *iz*, *jz*, *desc\_z*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading LOCp(*iz+n-1*) by LOCq(*jz+n-1*) part of the local array Z must contain the local pieces of the leading *iz+n-1* by *jz+n-1* part of the global matrix Z.

Scope: **local**

Returned as: an LLD\_Z by (at least) LOCq(N\_Z) array, containing numbers of the data type indicated in Table 116 on page 677.

*work* is the work area used by this subroutine if *lwork*  $\neq$  0, where:

If *lwork*  $\neq$  0 and *lwork*  $\neq$  -1, its size is (at least) of length *lwork*.

If *lwork* = -1, its size is (at least) of length 1.

Scope: **local**

Returned as: an area of storage, containing numbers of the data type indicated in Table 116 on page 677, where:

If *lwork*  $\geq$  1 or *lwork* = -1, then:

- If *jobz* = 'N', then *work*<sub>1</sub> is set to the minimum *lwork* needed.
- If *jobz* = 'V', then:

For PDSYEVX, *work*<sub>1</sub> is set to the minimum *lwork* needed to compute all eigenvectors, but not necessarily sufficient to guarantee orthogonality of the eigenvectors.

For PZHEEVX, *work*<sub>1</sub> is set to the minimum *lwork* needed to compute all eigenvectors.

- Except for *work*<sub>1</sub>, the contents of *work* are overwritten on return.

*rwork* is the work area used by this subroutine if *lrwork*  $\neq$  0, where:

If *lrwork*  $\neq$  0 and *lrwork*  $\neq$  -1, its size is (at least) of length *lrwork*.

If *lrwork* = -1, its size is (at least) of length 1.

Scope: **local**

Returned as: an area of storage, containing numbers of the data type indicated in Table 116 on page 677, where:

If *lrwork*  $\geq$  1 or *lrwork* = -1, then:

- If *jobz* = 'N', then *rwork*<sub>1</sub> is set to the minimum *lrwork* value needed.
- If *jobz* = 'V', then *rwork*<sub>1</sub> is set to the minimum *lrwork* value needed to compute all eigenvectors, but not necessarily sufficient to guarantee orthogonality of the eigenvectors.
- Except for *rwork*<sub>1</sub>, the contents of *rwork* are overwritten on return.

*iwork* is the work area used by this subroutine if *liwork*  $\neq$  0, where:

If *liwork*  $\neq$  0 and *liwork*  $\neq$  -1, then its size is (at least) of length *liwork*.

If *liwork* = -1, then its size is (at least) of length 1.

Scope: **local**

Returned as: an area of storage, where:

If *liwork*  $\geq$  1 or *liwork* = -1, then *iwork*<sub>1</sub> is set to the minimum *liwork* value and contains numbers of the data type indicated in Table 116 on page 677. Except for *iwork*<sub>1</sub>, the contents of *iwork* are overwritten on return.



*ifail* has the following meaning:

If *jobz* = 'N', then *ifail* is ignored.

If *jobz* = 'V', it is vector *ifail*, where:

- If there is a normal exit (see *info*), the first *m* elements of *ifail* are zero.
- If there is an error exit (where one or more eigenvectors failed to converge—see *info*), *ifail* contains the indices of the eigenvectors that failed to converge.

Scope: **global**

Returned as: a one-dimensional array of (at least) length *n*, containing fullword integers;  $0 \leq \text{ifail}_i \leq n$ .

*iclustr* has the following meaning:

If *jobz* = 'N', then *iclustr* is ignored.

If *jobz* = 'V', it is vector *iclustr*, containing the indices of the eigenvectors corresponding to a cluster of eigenvalues that could not be reorthogonalized due to insufficient workspace. Eigenvectors corresponding to clusters of eigenvalues indexed *iclustr*<sub>2i-1</sub> to *iclustr*<sub>2i</sub> could not be reorthogonalized due to lack of workspace. **Hence, the eigenvectors corresponding to these clusters may not be orthogonal.**

*iclustr* is a zero-terminated vector; that is, the last element of *iclustr* is set to zero. Assuming that *k* is the number of clusters, then:

- *iclustr*<sub>2k</sub> ≠ 0 and *iclustr*<sub>2k+1</sub> = 0

Scope: **global**

Returned as: a one-dimensional array of (at least) length 2(*nprow*)(*npcol*), containing fullword integers;  $0 \leq \text{iclustr}_i \leq n$ .

*gap* has the following meaning:

If *jobz* = 'N', then *gap* is ignored.

If *jobz* = 'V', it is vector *gap*, containing the gap between the eigenvalues whose eigenvectors could not be reorthogonalized. The values in this vector correspond to the clusters indicated by *iclustr*. As a result, the dot product between the eigenvectors corresponding to the *i*-th cluster may be as high as  $(C)(n)/\text{gap}_i$ , where *C* is a small constant.

Scope: **global**

Returned as: a one-dimensional array of (at least) length (*nprow*)(*npcol*), containing numbers of the data type indicated in Table 116 on page 677.

*info* has the following meaning:

If *info* = 0, then no input-argument errors or computational errors occurred. This indicates a normal exit.

**Note:** One use of *info* in ScaLAPACK is to identify whether input-argument errors occurred. Because Parallel ESSL terminates the application if input-argument errors occur, the setting of *info* is irrelevant for these errors.

If  $info > 0$ , then one or more of the following computational errors occurred and the appropriate error messages were issued, indicating an error exit, where:

- If  $\text{mod}(info, 2) \neq 0$ , then one or more eigenvectors failed to converge. Their indices are stored in *ifail*. (Ensure that  $abstol = (2)(unfl)$ .)
- If  $\text{mod}(info/2, 2) \neq 0$ , then the eigenvectors corresponding to one or more clusters of eigenvalues could not be reorthogonalized because of insufficient workspace. The indices of the clusters are stored in *iclustr*.
- If  $\text{mod}(info/4, 2) \neq 0$ , then all the eigenvectors between *vl* and *vu* could not be computed because of insufficient space. The number of eigenvectors computed is returned in *nz*.
- If  $\text{mod}(info/8, 2) \neq 0$ , then one or more eigenvalues were not computed. (Ensure that  $abstol = (2)(unfl)$ .)

Scope: **global**

Returned as: a fullword integer;  $info \geq 0$ .

## Notes and Coding Rules

1. This subroutine accepts lowercase letters for the *jobz*, *range*, and *uplo* arguments.
2. In your C program, argument *info* must be passed by reference.
3. *A*, *Z*, *w*, *ifail*, *iclustr*, *gap*, *work*, *rwork*, and *iwork* must have no common elements; otherwise, results are unpredictable.
4. The NUMROC utility subroutine can be used to determine the values of LOCp(M\_) and LOCq(N\_) used in the argument descriptions above. For details, see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28 and “NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process” on page 868.
5. The global matrix *A* must be distributed using a square block-cyclic distribution; that is,  $MB\_A = NB\_A$ .
6. The global matrix *A* must be aligned on a block boundary; that is:
  - $ia-1$  must be a multiple of  $MB\_A$
  - $ja-1$  must be a multiple of  $NB\_A$
7. If *jobz* = 'V', then also follow these rules:
  - In the process grid, the process row containing the first row of the submatrix *A* must also contain the first row of the submatrix *Z*; that is:  $iarow = izrow$
  - where:
    - $iarow = \text{mod}(\text{RSRC\_A} + (ia-1)/MB\_A, p)$
    - $izrow = \text{mod}(\text{RSRC\_Z} + (iz-1)/MB\_Z, p)$
  - $M\_A = M\_Z$
  - $MB\_A = MB\_Z$
  - $NB\_A = NB\_Z$
  - $\text{RSRC\_A} = \text{RSRC\_Z}$
  - $\text{CSRC\_A} = \text{CSRC\_Z}$
  - $\text{CTXT\_A} = \text{CTXT\_Z}$
  - The block row offset of the global matrix *A* must be equal to the block row offset of the global general matrix *Z*; that is:
    - $\text{mod}((ia-1, MB\_A) = \text{mod}(iz-1, MB\_Z))$

8. Eigenvectors associated with tightly clustered eigenvalues may not be orthogonal.
9. Eigenvectors that are on different processes are not reorthogonalized. For details, see the *lwork* and *lrwork* arguments.
10. An example of the use of PDSYEVX in a thermal diffusion application program is shown in Appendix B, "Sample Programs." See "Program Main" on page 902.
11. If *lwork* = -1 on any process, it must equal -1 on all processes. That is, if a subset of the processes specifies -1 for the work area size, they must all specify -1.
12. If *lrwork* = -1 on any process, it must equal -1 on all processes. That is, if a subset of the processes specifies -1 for the work area size, they must all specify -1.
13. If *liwork* = -1 on any process, it must equal -1 on all processes. That is, if a subset of the processes specifies -1 for the work area size, they must all specify -1.
14. The following describes the relationship between workspace, orthogonality, and performance.  
Let *clustersize* be the number of eigenvalues in the largest cluster, where a cluster is defined as a set of close eigenvalues:
  - $\{w_k, \dots, w_{k+clustersize-1} \mid w_{j+1} \leq w_j + orfac(2)(\text{norm}(A))\}$
  - If *clustersize* is:

$$clustersize \geq n / \sqrt{(nprow)(npcol)}$$

then providing enough space to compute all the eigenvectors orthogonally causes serious degradation in performance. In the limit (*clustersize* = *n*-1), performance may be no better than using one process.

- If *clustersize* is:

$$clustersize = n / \sqrt{(nprow)(npcol)}$$

then reorthogonalizing all eigenvectors increases the total execution time by a factor of 2 or more.

- If *clustersize* is:

$$clustersize > n / \sqrt{(nprow)(npcol)}$$

then execution time grows as the square of the cluster size, assuming all other factors remain equal and there is enough workspace. Less workspace means less reorthogonalization, but faster execution.

For PDSYEVX, see the description of *work* on page 681. For PZHEEVX, see the description of *rwork* on page 683.

## Function

This subroutine computes selected eigenvalues and, optionally, the eigenvectors of a real symmetric or complex Hermitian matrix *A*. Eigenvalues and eigenvectors

can be selected by specifying a range of values or a range of indices for the eigenvalues. The computation involves the following steps:

1. Reduce the matrix  $A$  to real symmetric tridiagonal form.
2. Compute the requested eigenvalues of the real symmetric tridiagonal matrix using bisection.
3. If requested, compute the eigenvectors of the real symmetric tridiagonal matrix using inverse iteration, and then back transform the eigenvectors to obtain the eigenvectors of the matrix  $A$ .

## Error Conditions

### Computational Errors

**Note:** For more details, see output argument *info*.

1. Bisection failed to converge for some eigenvalues. The eigenvalues may not be as accurate as the absolute and relative tolerances.
2. The number of eigenvalues computed does not match the number of eigenvalues requested.
3. No eigenvalues were computed, because the Gershgorin interval initially used was incorrect.
4. Some eigenvectors failed to converge. The indices are stored in *ifail*.
5. Eigenvectors corresponding to one or more clusters of eigenvalues could not be reorthogonalized because of insufficient workspace. The indices of the clusters are stored in *iclustr*.
6. All the eigenvectors between  $vl$  and  $vu$  could not be computed due to insufficient workspace. The number of eigenvectors computed is returned in  $nz$ .

### Resource Errors

1. ( $lwork = 0$ ,  $lrwork = 0$ , or  $liwork = 0$ ) and unable to allocate work space

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1.  $DTYPE\_A$  is invalid.
2.  $DTYPE\_Z$  is invalid and  $jobz = 'V'$

#### Stage 2:

1.  $CTXT\_A$  is invalid.

#### Stage 3:

1. This subroutine has been called from outside the process grid.

#### Stage 4:

1.  $jobz \neq 'N'$  or  $'V'$
2.  $range \neq 'A'$ ,  $'V'$ , or  $'T'$
3.  $uplo \neq 'U'$  or  $'L'$
4.  $n < 0$
5.  $M\_A < 0$  and  $n = 0$ ;  $M\_A < 1$  otherwise
6.  $N\_A < 0$  and  $n = 0$ ;  $N\_A < 1$  otherwise
7.  $MB\_A < 1$
8.  $NB\_A < 1$
9.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
10.  $CSRC\_A < 0$  or  $CSRC\_A \geq q$
11.  $ia < 1$

12.  $ja < 1$   
If  $jobz = 'V'$ :
13.  $M\_Z < 0$  and  $n = 0$ ;  $M\_Z < 1$  otherwise
14.  $N\_Z < 0$  and  $n = 0$ ;  $N\_Z < 1$  otherwise
15.  $MB\_Z < 1$
16.  $NB\_Z < 1$
17.  $RSRC\_Z < 0$  or  $RSRC\_Z \geq p$
18.  $CSRC\_Z < 0$  or  $CSRC\_Z \geq q$
19.  $iz < 1$
20.  $jz < 1$
21.  $CTXT\_A \neq CTXT\_Z$

**Stage 5:**

1.  $vu \leq vl$  and  $range = 'V'$  and  $n \neq 0$
2.  $il < 1$  and  $range = 'T'$
3.  $(iu < \min(n, il)$  or  $iu > n)$  and  $range = 'T'$   
If  $n \neq 0$ :
4.  $ia > M\_A$
5.  $ja > N\_A$
6.  $ia+n-1 > M\_A$
7.  $ja+n-1 > N\_A$   
If  $n \neq 0$  and  $jobz = 'V'$ :
8.  $iz > M\_Z$
9.  $jz > N\_Z$
10.  $iz+n-1 > M\_Z$
11.  $jz+n-1 > N\_Z$   
In all cases:
12.  $MB\_A \neq NB\_A$
13.  $\text{mod}(ia-1, MB\_A) \neq 0$
14.  $\text{mod}(ja-1, NB\_A) \neq 0$   
If  $jobz = 'V'$ :
15.  $M\_A \neq M\_Z$
16.  $MB\_A \neq MB\_Z$
17.  $NB\_A \neq NB\_Z$
18.  $\text{mod}(iz-1, MB\_Z) \neq \text{mod}(ia-1, MB\_A)$
19. In the process grid, the process row containing the first row of the submatrix  $A$  does not contain the first row of the submatrix  $Z$ ; that is,  $iarrow \neq izrow$ , where:
  - a.  $iarrow = \text{mod}(RSRC\_A + (ia-1)/MB\_A, p)$
  - b.  $izrow = \text{mod}(RSRC\_Z + (iz-1)/MB\_Z, p)$
20.  $RSRC\_A \neq RSRC\_Z$
21.  $CSRC\_A \neq CSRC\_Z$

**Stage 6:**

1.  $LLD\_A < \max(1, \text{LOCp}(M\_A))$
2.  $lwork \neq 0$ ,  $lwork \neq -1$ , and  $lwork < (\text{minimum value})$ . (For the minimum value, see the  $lwork$  argument description.)
3.  $lrwork \neq 0$ ,  $lrwork \neq -1$ , and  $lrwork < (\text{minimum value})$ . (For the minimum value, see the  $lrwork$  argument description.)
4.  $liwork \neq 0$ ,  $liwork \neq -1$ , and  $liwork < (\text{minimum value})$ . (For the minimum value, see the  $liwork$  argument description.)  
If  $jobz = 'V'$ :
5.  $LLD\_Z < \max(1, \text{LOCp}(M\_Z))$

**Stage 7:**

Each of the following global input arguments are checked to determine whether its value differs from the value specified on process  $P_{00}$ :

1. *jobz* differs.
2. *range* differs.
3. *uplo* differs.
4. *n* differs.
5. *ia* differs.
6. *ja* differs.
7. *DTYPE\_A* differs.
8. *M\_A* differs.
9. *N\_A* differs.
10. *MB\_A* differs.
11. *NB\_A* differs.
12. *RSRC\_A* differs.
13. *CSRC\_A* differs.
14. *ABSTOL* differs.

Also:

15. *lwork* = -1 on a subset of processes.
16. *lrwork* = -1 on a subset of processes.
17. *liwork* = -1 on a subset of processes.

### Stage 8:

Each of the following global input arguments are checked to determine whether its value differs from the value specified on process  $P_{00}$ :

If *range* = 'V':

1. *vl* differs.
2. *vu* differs.

If *range* = 'T':

3. *il* differs.
4. *iu* differs.

If *jobz* = 'V':

5. *iz* differs.
6. *jz* differs.
7. *DTYPE\_Z* differs.
8. *M\_Z* differs.
9. *N\_Z* differs.
10. *MB\_Z* differs.
11. *NB\_Z* differs.
12. *RSRC\_Z* differs.
13. *CSRC\_Z* differs.
14. *ORFAC* differs.

## Examples

### Example 1

This example shows how to find all the eigenvalues and eigenvectors of a real symmetric matrix *A* of order 4 using a  $2 \times 2$  process grid.

#### Notes:

1. Because *range* = 'A', arguments *vl*, *vu*, *il*, and *iu* are not referenced.
2. Because *lwork* = 0 and *liwork* = 0, PDSYEVX dynamically allocates the work areas used by this subroutine.

## Call Statements and Input:

```

ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      JOBZ RANGE UPLO  N  A  IA  JA  DESC_A  VL  VU  IL  IU  ABSTOL  M  NZ  W
CALL PDSYEVX( 'V', 'A', 'U', 4, A, 1, 1, DESC_A, 0.0D0, 0.0D0, 0, 0, -1.0D0, M, NZ, W,

+
      ORFAC  Z  IZ  JZ  DESC_Z  WORK  LWORK  IWORK  LIWORK  IFAIL  ICLUSTER  GAP  INFO
      -1.0D0, Z, 1, 1, DESC_Z, WORK, 0, IWORK, 0, IFAIL, ICLUSTER, GAP, INFO)

```

	DESC_A	DESC_Z
DTYPE_	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	4	4
N_	4	4
MB_	1	1
NB_	1	1
RSRC_	0	0
CSRC_	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>

## Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:

```

LLD_A = MAX(1, NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_Z = MAX(1, NUMROC(M_Z, MB_Z, MYROW, RSRC_Z, NPROW))

```

In this example,  $LLD_A = LLD_Z = 2$  on all processes.

Global real symmetric matrix *A* of order 4 with block sizes  $1 \times 1$ :

B,D	0	1	2	3
0	5.0	4.0	1.0	1.0
1	.	5.0	1.0	1.0
2	.	.	4.0	2.0
3	.	.	.	4.0

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$

Local arrays for  $A$ :

p,q	0		1	
0	5.0	1.0	4.0	1.0
	.	4.0	.	2.0
1	.	1.0	5.0	1.0
	.	.	.	4.0

**Output:**

The upper triangle, including the diagonal, of the global real symmetric matrix  $A$  is overwritten; that is, the original input is not preserved.

On all processes,  $m = 4$  and  $nz = 4$ .

Global vector  $w$  of length 4 is the same on all processes:

- $w = (1.00, 2.00, 5.00, 10.00)$

Global general matrix  $Z$  of order 4 with block sizes  $1 \times 1$ :

B,D	0	1	2	3
0	0.7071	0.0000	-0.3162	-0.6325
1	-0.7071	0.0000	-0.3162	-0.6325
2	0.0000	-0.7071	0.6325	-0.3162
3	0.0000	0.7071	0.6325	-0.3162

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$
3		

Local arrays for  $Z$ :

p,q	0		1	
0	0.7071	-0.3162	0.0000	-0.6325
	0.0000	0.6325	-0.7071	-0.3162
1	-0.7071	-0.3162	0.0000	-0.6325
	0.0000	0.6325	0.7071	-0.3162

Global vector *ifail* of length 4 is the same on all processes:

- *ifail* = (0, 0, 0, 0)

Global vector *iclustr* of length 8 ( =  $2(nprow)(npcol)$ ) is the same on all processes:

- *iclustr* = (0, 0, 0, 0, 0, 0, 0, 0)

Global vector *gap* of length 4 ( =  $(nprow)(npcol)$ ) is the same on all processes:

- *gap* = (-1.0, -1.0, -1.0, -1.0)

The value of *info* is 0 on all processes.



## Example 2

This example shows how to find all the eigenvalues and eigenvectors of a complex Hermitian matrix  $A$  of order 4 using a  $2 \times 2$  process grid.

### Notes:

1. Because  $range = 'A'$ , arguments  $vl$ ,  $vu$ ,  $il$ , and  $iu$  are not referenced.
2. Because  $lwork = 0$ ,  $lrwork = 0$ , and  $liwork = 0$ , PZHEEVX dynamically allocates the work areas used by this subroutine.

### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      JOBZ RANGE UPLO  N  A  IA  JA  DESC_A  VL  VU  IL  IU  ABSTOL  M  NZ  W  ORFAC
CALL PZHEEVX( 'V',  'A',  'L',  4,  A,  1,  1,  DESC_A,  0.0D0,  0.0D0,  0,  0, -1.0D0,  M,  NZ,  W, -1.0D0,

+
      Z  IZ  JZ  DESC_Z  WORK  LWORK  RWORK  LRWORK  IWORK  LIWORK  IFAIL  ICLUSTER  GAP  INFO
      Z,  1,  1,  DESC_Z,  WORK ,  0,  RWORK,  0,  IWORK ,  0,  IFAIL,  ICLUSTER,  GAP,  INFO)
```

	DESC_A	DESC_Z
DTYPE_	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	4	4
N_	4	4
MB_	1	1
NB_	1	1
RSRC_	0	0
CSRC_	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>

### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:  
 $LLD\_A = \text{MAX}(1, \text{NUMROC}(M\_A, MB\_A, MYROW, RSRC\_A, NPROW))$   
 $LLD\_Z = \text{MAX}(1, \text{NUMROC}(M\_Z, MB\_Z, MYROW, RSRC\_Z, NPROW))$   
 In this example,  $LLD\_A = LLD\_Z = 2$  on all processes.

Global complex Hermitian matrix  $A$  of order 4 with block sizes  $1 \times 1$ :

B,D	0	1	2	3
0	(5.0, 0.0)	.	.	.
1	(4.0, 1.0)	(5.0, 0.0)	.	.
2	(1.0, 2.0)	(1.0, 0.0)	(4.0, 0.0)	.
3	(2.0, 3.0)	(3.0, 2.0)	(5.0, 1.0)	(4.0, 0.0)

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$
3		

Local arrays for  $A$ :

p,q	0	1
0	(5.0, . ) (1.0, 2.0) (4.0, . )	(1.0, 0.0) .
1	(4.0, 1.0) . (2.3, 3.0) (5.0, 1.0)	(5.0, . ) (3.0, 2.0) (4.0, . )

### Output:

The lower triangle, including the diagonal, of the global complex Hermitian matrix  $A$  is overwritten; that is, the original input is not preserved.

On all processes,  $m = 4$  and  $nz = 4$ .

Global vector  $w$  of length 4 is the same on all processes:

- $w = (-1.765, 0.727, 4.664, 14.374)$

Global general matrix  $Z$  of order 4 with block sizes  $1 \times 1$ :

B,D	0	1	2	3
0	( .0926, .0000)	( .7591, .0000)	( .3758, .0000)	(-.5234, .0000)
1	( .0840,-.2873)	(-.5874, .0177)	( .5370,-.2067)	(-.4515,-.1735)
2	( .5013,-.3461)	( .0526,-.0674)	(-.6287,-.2149)	(-.2864,-.3134)
3	(-.6703, .2854)	(-.0155,-.2662)	(-.2294,-.1834)	(-.3059,-.4672)

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$
3		

Local arrays for  $Z$ :

p,q	0	1
0	( .0926, .0000) ( .3758, .0000) ( .5013,-.3461) (-.6287,-.2149)	( .7591, .0000) (-.5234, .0000) ( .0526,-.0674) (-.2864,-.3134)
1	( .0840,-.2873) ( .5370,-.2067) (-.6703, .2854) (-.2294,-.1834)	(-.5874, .0177) (-.4515,-.1735) (-.0155,-.2662) (-.3059,-.4672)

Global vector *ifail* of length 4 is the same on all processes:

- *ifail* = (0, 0, 0, 0)

Global vector *iclustr* of length 8 ( =  $2(nprow)(npcol)$ ) is the same on all processes:

- *iclustr* = (0, 0, 0, 0, 0, 0, 0, 0)

Global vector *gap* of length 4 ( =  $(nprow)(npcol)$ ) is the same on all processes:

- *gap* = (-1.0, -1.0, -1.0, -1.0)

The value of *info* is 0 on all processes.

## PDSYGVX and PZHEGVX — Selected Eigenvalues and, Optionally, the Eigenvectors of a Real Symmetric or Complex Hermitian Positive Definite Generalized Eigenproblem

### Purpose

These subroutines compute selected eigenvalues and, optionally, the eigenvectors of a real symmetric or complex Hermitian positive definite generalized eigenproblem:

- If  $ibtype = 1$ , the problem is  $Ax = \lambda Bx$
- If  $ibtype = 2$ , the problem is  $ABx = \lambda x$
- If  $ibtype = 3$ , the problem is  $BAx = \lambda x$

In the formulas above:

- $A$  represents the global real symmetric or complex Hermitian submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$
- $B$  represents the global real symmetric or complex Hermitian positive definite submatrix  $B_{ib:ib+n-1, jb:jb+n-1}$

Eigenvalues and eigenvectors can be selected by specifying a range of values or a range of indices for the desired eigenvalues.

If  $n = 0$ , no computation is performed and the subroutine returns after doing some parameter checking.

See references [13], [25], [26], and [27].

Table 117. Data Types

<i>vl, vu, abstol, w, orfac, rwork, gap</i>	<i>A, B, Z, work</i>	<i>iwork, ifail, iclustr</i>	Subroutine
Long-precision real	Long-precision real	Integer	PDSYGVX
Long-precision real	Long-precision complex	Integer	PZHEGVX

### Syntax

Fortran	CALL PDSYGVX ( <i>ibtype, jobz, range, uplo, n, a, ia, ja, desc_a, b, ib, jb, desc_b, vl, vu, il, iu, abstol, m, nz, w, orfac, z, iz, jz, desc_z, work, lwork, iwork, liwork, ifail, iclustr, gap, info</i> )
	CALL PZHEGVX ( <i>ibtype, jobz, range, uplo, n, a, ia, ja, desc_a, b, ib, jb, desc_b, vl, vu, il, iu, abstol, m, nz, w, orfac, z, iz, jz, desc_z, work, lwork, rwork, lrwork, iwork, liwork, ifail, iclustr, gap, info</i> )
C and C++	pdsygvx ( <i>ibtype, jobz, range, uplo, n, a, ia, ja, desc_a, b, ib, jb, desc_b, vl, vu, il, iu, abstol, m, nz, w, orfac, z, iz, jz, desc_z, work, lwork, iwork, liwork, ifail, iclustr, gap, info</i> );
	pzhgvx ( <i>ibtype, jobz, range, uplo, n, a, ia, ja, desc_a, b, ib, jb, desc_b, vl, vu, il, iu, abstol, m, nz, w, orfac, z, iz, jz, desc_z, work, lwork, rwork, lrwork, iwork, liwork, ifail, iclustr, gap, info</i> );

### On Entry

*ibtype* specifies the problem type, where:

- If  $ibtype = 1$ , the problem is  $Ax = \lambda Bx$
- If  $ibtype = 2$ , the problem is  $ABx = \lambda x$
- If  $ibtype = 3$ , the problem is  $BAx = \lambda x$

Scope: **global**

Specified as: a fullword integer; *ibtype* = 1, 2, or 3.

*jobz* indicates the type of computation to be performed, where:

If *jobz* = 'N', eigenvalues only are computed.

If *jobz* = 'V', eigenvalues and eigenvectors are computed.

Scope: **global**

Specified as: a single character; *jobz* = 'N' or 'V'.

*range* indicates which eigenvalues to compute, where:

If *range* = 'A', all eigenvalues are to be found.

If *range* = 'V', all eigenvalues in the interval [*vl*, *vu*] are to be found.

If *range* = 'T', the *il*-th through *iu*-th eigenvalues are to be found.

Scope: **global**

Specified as: a single character; *range* = 'A', 'V', or 'T'.

*uplo* indicates whether the upper or lower triangular part of the global submatrices *A* and *B* are referenced, where:

If *uplo* = 'U', the upper triangular part is referenced.

If *uplo* = 'L', the lower triangular part is referenced.

Scope: **global**

Specified as: a single character; *uplo* = 'U' or 'L'.

*n* is the order of submatrices *A* and *B* used in the computation.

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

*a* is the local part of the global real symmetric or complex Hermitian matrix *A*. This identifies the **first element** of the local array *A*. This subroutine computes the location of the first element of the local subarray used, based on *ia*, *ja*, *desc\_a*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading LOCp(*ia*+*n*-1) by LOCq(*ja*+*n*-1) part of the local array *A* must contain the local pieces of the leading *ia*+*n*-1 by *ja*+*n*-1 part of the global matrix, and:

- If *uplo* = 'U', the leading  $n \times n$  upper triangular part of the global submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  must contain the upper triangular part of the matrix, and the strictly lower triangular part is not referenced.
- If *uplo* = 'L', the leading  $n \times n$  lower triangular part of the global submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  must contain the lower triangular part of the matrix, and the strictly upper triangular part is not referenced.

Scope: **local**

Specified as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 117 on page 698. Details about the square block-cyclic data distribution of global matrix *A* are stored in *desc\_a*.

*ia* is the row index of the global matrix *A*, identifying the first row of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq M\_A$  and  $ia+n-1 \leq M\_A$ .

## PDSYGVX and PZHEGVX

*ja* is the column index of the global matrix *A*, identifying the first column of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$  and  $ja+n-1 \leq N\_A$ .

*desc\_a* is the array descriptor for global matrix *A*, described in the following table:

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_A < p$	Global
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < q$	Global
9	LLD_A	The leading dimension of the local array	$LLD\_A \geq \max(1, LOCp(M\_A))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

*b* is the local part of the global real symmetric or complex Hermitian positive definite matrix *B*. This identifies the **first element** of the local array *B*. This subroutine computes the location of the first element of the local subarray used, based on *ib*, *jb*, *desc\_b*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading  $LOCp(ib+n-1)$  by  $LOCq(jb+n-1)$  part of the local array *B* must contain the local pieces of the leading  $ib+n-1$  by  $jb+n-1$  part of the global matrix, and:

- If *uplo* = 'U', the leading  $n \times n$  upper triangular part of the global submatrix  $B_{ib:ib+n-1, jb:jb+n-1}$  must contain the upper triangular part of the matrix, and the strictly lower triangular part is not referenced.
- If *uplo* = 'L', the leading  $n \times n$  lower triangular part of the global submatrix  $B_{ib:ib+n-1, jb:jb+n-1}$  must contain the lower triangular part of the matrix, and the strictly upper triangular part is not referenced.

Scope: **local**

Specified as: an LLD\_B by (at least)  $LOCq(N\_B)$  array, containing numbers of the data type indicated in Table 119 on page 739. Details about the square block-cyclic data distribution of global matrix *B* are stored in *desc\_b*.

*ib* is the row index of the global matrix *B*, identifying the first row of the submatrix *B*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ib \leq M\_B$  and  $ib+n-1 \leq M\_B$ .

*jb* is the column index of the global matrix *B*, identifying the first column of the submatrix *B*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq jb \leq N\_B$  and  $jb+n-1 \leq N\_B$ .

*desc\_b* is the array descriptor for global matrix *B*, described in the following table:

<i>desc_b</i>	Name	Description	Limits	Scope
1	DTYPE_B	Descriptor type	DTYPE_B=1	Global
2	CTXT_B	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_B	Number of rows in the global matrix	If $n = 0$ : $M\_B \geq 0$ Otherwise: $M\_B \geq 1$	Global
4	N_B	Number of columns in the global matrix	If $n = 0$ : $N\_B \geq 0$ Otherwise: $N\_B \geq 1$	Global
5	MB_B	Row block size	$MB\_B \geq 1$	Global
6	NB_B	Column block size	$NB\_B \geq 1$	Global
7	RSRC_B	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_B < p$	Global
8	CSRC_B	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_B < q$	Global
9	LLD_B	The leading dimension of the local array	$LLD\_B \geq \max(1, LOCp(M\_B))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

*vl* has the following meaning:

If *range* = 'V', it is the lower bound of the interval to be searched for eigenvalues.

If *range*  $\neq$  'V', this argument is ignored.

Scope: **global**

Specified as: a number of the data type indicated in Table 117 on page 698.

If *range* = 'V',  $vl < vu$ .

*vu* has the following meaning:

If *range* = 'V', it is the upper bound of the interval to be searched for eigenvalues.

If *range*  $\neq$  'V', this argument is ignored.

Scope: **global**

Specified as: a number of the data type indicated in Table 117 on page 698.

If *range* = 'V',  $vl < vu$ .

*il* has the following meaning:

If  $range = 'T'$ , it is the index (from smallest to largest) of the smallest eigenvalue to be returned.

If  $range \neq 'T'$ , this argument is ignored.

Scope: **global**

Specified as: a fullword integer;  $il \geq 1$ .

*iu* has the following meaning:

If  $range = 'T'$ , it is the index (from smallest to largest) of the largest eigenvalue to be returned.

If  $range \neq 'T'$ , this argument is ignored.

Scope: **global**

Specified as: a fullword integer;  $\min(il, n) \leq iu \leq n$ .

*abstol* is the absolute tolerance for the eigenvalues. An approximate eigenvalue is accepted as converged when it is determined to lie in an interval  $[a, b]$  of width less than or equal to:

- $abstol + \epsilon(\max(|a|, |b|))$

where  $\epsilon$  is the machine precision. If *abstol* is less than or equal to zero, then  $\epsilon(\text{norm}(T))$  is used in its place, where  $\text{norm}(T)$  is the 1-norm of the tridiagonal matrix obtained by reducing the standard form of the generalized problem to tridiagonal form. For most problems, this is the appropriate level of accuracy to request.

For certain strongly graded matrices, greater accuracy can be obtained in very small eigenvalues by setting *abstol* to a very small positive number. However, if *abstol* is less than:

$$\sqrt{unfl}$$

where *unfl* is the underflow threshold, then:

$$\sqrt{unfl}$$

is used in its place.

Eigenvalues are computed most accurately when *abstol* is set to twice the underflow threshold—that is,  $(2)(unfl)$ .

If  $jobz = 'V'$ , then setting *abstol* to *unfl*, the underflow threshold, yields the most orthogonal eigenvectors.



**Note:**

- $\varepsilon$  is approximately  $0.222044604925031308\text{E}-15$
- $unfl$  is approximately  $0.222507385850720138\text{E}-307$
- $\sqrt{unfl}$  is approximately  $0.149166814624004135\text{E}-153$

Scope: **global**

Specified as: a number of the data type indicated in Table 117 on page 698.

$m$  See On Return.

$nz$  See On Return.

$w$  See On Return.

$orfac$  specifies which eigenvectors should be reorthogonalized. Eigenvectors that correspond to eigenvalues which are within:

- $ortol = (orfac)(\text{norm}(\mathbf{R}))$

of each other (where  $\text{norm}(\mathbf{R})$  is the 1-norm of the standard form of the generalized eigenproblem) are to be reorthogonalized.

However, if the workspace is insufficient (see  $lwork$  and  $lrwork$ ),  $ortol$  may be decreased until all eigenvectors to be reorthogonalized can be stored in one process.

If  $orfac$  is zero, no reorthogonalization is done.

If  $orfac$  is less than zero, a default value of  $10^{-3}$  is used.

Scope: **global**

Specified as: a number of the data type indicated in Table 117 on page 698.

$z$  See On Return.

$iz$  is the row index of the global matrix  $\mathbf{Z}$ , identifying the first row of the submatrix  $\mathbf{Z}$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq iz \leq M\_Z$  and  $iz+n-1 \leq M\_Z$ .

$jz$  is the column index of the global matrix  $\mathbf{Z}$ , identifying the first column of the submatrix  $\mathbf{Z}$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq jz \leq N\_Z$  and  $jz+n-1 \leq N\_Z$ .

$desc\_z$  is the array descriptor for global matrix  $\mathbf{Z}$ , described in the following table:

$desc\_z$	Name	Description	Limits	Scope
1	DTYPE_Z	Descriptor type	DTYPE_Z=1	Global
2	CTXT_Z	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global

## PDSYGVX and PZHEGVX

<i>desc_z</i>	Name	Description	Limits	Scope
3	M_Z	Number of rows in the global matrix	If $n = 0$ : $M\_Z \geq 0$ Otherwise: $M\_Z \geq 1$	Global
4	N_Z	Number of columns in the global matrix	If $n = 0$ : $N\_Z \geq 0$ Otherwise: $N\_Z \geq 1$	Global
5	MB_Z	Row block size	$MB\_Z \geq 1$	Global
6	NB_Z	Column block size	$NB\_Z \geq 1$	Global
7	RSRC_Z	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_Z < p$	Global
8	CSRC_Z	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_Z < q$	Global
9	LLD_Z	The leading dimension of the local array	$LLD\_Z \geq \max(1, LOCp(M\_Z))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

*work* has the following meaning:

If  $lwork = 0$ , *work* is ignored.

If  $lwork \neq 0$ , *work* is a work area used by this subroutine, where:

- If  $lwork \neq -1$ , then its size is (at least) of length *lwork*.
- If  $lwork = -1$ , then its size is (at least) of length 1.

Scope: **local**

Specified as: an area of storage containing numbers of data type indicated in Table 117 on page 698.

*lwork* is the number of elements in array WORK.

Scope:

- If  $lwork \geq 0$ , *lwork* is **local**.
- If  $lwork = -1$ , *lwork* is **global**.

Specified as: a fullword integer; where:

- If  $lwork = 0$ , PDSYGVX and PZHEGVX dynamically allocate the work area used by the subroutine. The work area is deallocated before control is returned to the calling program. This option is an extension to the ScaLAPACK standard.
- If  $lwork = -1$ , PDSYGVX and PZHEGVX perform a work area query and return the minimum required size of *work* in *work<sub>1</sub>*. No computation is performed and the subroutine returns after error checking is complete.
- Otherwise, use the following rules to determine the value to specify:
  - If *jobz* = 'N', it must have the following value:  
For PDSYGVX,  $lwork \geq 3(n+ja-1)+2n + \max(5nn, (nb)(np+1))$   
For PZHEGVX,  $lwork \geq n+ja-1 + \max(3nb, nb(np+1))$
  - If *jobz* = 'V', then the amount of workspace required to guarantee that all eigenvectors are computed is:

For PDSYGVX,  $lwork \geq 3(n+ja-1)+2n + \max(5nn, (np0)(mq0)+2(nb)(nb)) + \text{iceil}(neig, (nprow)(npcol))(nn)$

For PZHEGVX,  $lwork \geq n+ja-1 + (np0+mq0+nb)(nb)$

where:

- $nn = \max(n, nb, 2)$
- $neig$  is the number of eigenvectors requested
- $nb = MB\_A = NB\_A = MB\_Z = NB\_Z$
- $np = \text{NUMROC}(nn, nb, myrow, iarow, nprow)$
- $np0 = \text{NUMROC}(nn+iroffz, nb, izrow, izrow, nprow)$
- $mq0 = \text{NUMROC}(\max(neig, nb, 2)+icoffz, nb, izcol, izcol, npc0)$
- $iarow = \text{mod}(\text{RSRC\_A} + (ia-1)/nb, nprow)$
- $izrow = \text{mod}(\text{RSRC\_Z} + (iz-1)/nb, nprow)$
- $izcol = \text{mod}(\text{CSRC\_Z} + (jz-1)/nb, npc0)$
- $iroffz = \text{mod}(iz-1, MB\_Z)$
- $icoffz = \text{mod}(jz-1, NB\_Z)$

For PDSYGVX, the computed eigenvectors may not be orthogonal if the minimum workspace is supplied and *ortol* is too small; therefore, if you want to guarantee orthogonality (at the cost of potentially compromising performance), you should add the following to *lwork*:

$(clustersize-1)(n)$

where *clustersize* is the number of eigenvalues in the largest cluster, where a cluster is defined as a set of close eigenvalues:

- $\{w_k, \dots, w_{k+iclustrsz-1} \mid w_{j+1} \leq w_j + \text{orfac}(2)(\text{norm}(A))\}$

**Note:** PDSYGVX does **not** add this amount when dynamically allocating this workspace. You must use static allocation if you want to guarantee orthogonality.

When *lwork* is too small:

- For PDSYGVX, if *lwork* is too small to guarantee orthogonality, this subroutine attempts to maintain orthogonality in the clusters with the smallest spacing between the eigenvalues.
- If *lwork* is too small to compute all the eigenvectors requested, no computation is performed, except that if *range* = 'V', this subroutine does not know how many eigenvectors are requested until the eigenvalues are computed. Therefore, if *range* = 'V' and *lwork* is large enough to allow this subroutine to compute the eigenvalues, this subroutine computes the eigenvalues and as many eigenvectors as it can.

For the relationship between workspace, orthogonality, and performance, see Notes and Coding Rules 18 on page 712.

*rwork* has the following meaning:

If *lrwork* = 0, *rwork* is ignored.

If *lrwork*  $\neq$  0, *rwork* is a work area used by this subroutine, where:

- If *lrwork*  $\neq$  -1, then its size is (at least) of length *lrwork*.
- If *lrwork* = -1, then its size is (at least) of length 1.

Scope: **local**

Specified as: an area of storage containing numbers of data type indicated in Table 117 on page 698.

*lrwork* is the number of elements in array RWORK.

Scope:

- If  $lwork \geq 0$ ,  $lwork$  is **local**.
- If  $lwork = -1$ ,  $lwork$  is **global**.

Specified as: a fullword integer; where:

- If  $lwork = 0$ , PZHEGVX dynamically allocates the work area used by this subroutine. The work area is deallocated before control is returned to the calling program. This option is an extension to the ScaLAPACK standard.
- If  $lwork = -1$ , PZHEGVX performs a work area query and return the minimum required size of  $rwork$  in  $rwork_1$ . No computation is performed and the subroutine returns after error checking is complete.
- Otherwise, use the following rules to determine the value to specify:
  - If  $jobz = 'N'$ , it must have the following value:  

$$lwork \geq 2(n+ja-1) + 2n + 5nn$$
  - If  $jobz = 'V'$ , then the amount of workspace required to guarantee that all eigenvectors are computed is:  

$$lwork \geq 2(n+ja-1) + 2n + \max(5nn, (np0)(mq0)) + \text{iceil}(neig, (nprow)(npcol))(nn)$$

where:

- $nn = \max(n, nb, 2)$
- $neig$  is the number of eigenvectors requested
- $nb = MB\_A = NB\_A = MB\_Z = NB\_Z$
- $np0 = \text{NUMROC}(nn+iroffz, nb, izrow, izrow, nprow)$
- $mq0 = \text{NUMROC}(\max(neig, nb, 2)+icoffz, nb, izcol, izcol, npc0)$
- $izrow = \text{mod}(\text{RSRC\_Z} + (iz-1)/nb, nprow)$
- $izcol = \text{mod}(\text{CSRC\_Z} + (jz-1)/nb, npc0)$
- $iroffz = \text{mod}(iz-1, MB\_Z)$
- $icoffz = \text{mod}(jz-1, NB\_Z)$

For PZHEGVX, the computed eigenvectors may not be orthogonal if the minimum workspace is supplied and  $ortol$  is too small; therefore, if you want to guarantee orthogonality (at the cost of potentially compromising performance), you should add the following to  $lwork$ :

$(clustersize-1)(n)$

where  $clustersize$  is the number of eigenvalues in the largest cluster, where a cluster is defined as a set of close eigenvalues:

- $\{w_{k'}, \dots, w_{k+iclustrsz-1} \mid w_{j+1} \leq w_j + \text{orfac}(2)(\text{norm}(A))\}$

**Note:** PZHEGVX does **not** add this amount when dynamically allocating this workspace. You must use static allocation if you want to guarantee orthogonality.

When  $lwork$  is too small:

- If  $lwork$  is too small to guarantee orthogonality, this subroutine attempts to maintain orthogonality in the clusters with the smallest spacing between the eigenvalues.
- If  $lwork$  is too small to compute all the eigenvectors requested, no computation is performed, except that if  $range = 'V'$ , this subroutine does not know how many eigenvectors are requested until the eigenvalues are computed. Therefore, if  $range = 'V'$  and  $lwork$  is large enough to allow this subroutine to compute the eigenvalues, this subroutine computes the eigenvalues and as many eigenvectors as it can.

For the relationship between workspace, orthogonality, and performance, see Notes and Coding Rules 18 on page 712.

*iwork* has the following meaning:

If *liwork* = 0, *iwork* is ignored.

If *liwork* ≠ 0, *iwork* is a work area used by this subroutine, where:

- If *liwork* ≠ -1, then its size is (at least) of length *liwork*.
- If *liwork* = -1, then its size is (at least) of length 1.

Scope: **local**

Specified as: an area of storage containing numbers of data type indicated in Table 117 on page 698.

*liwork* is the number of elements in array IWORK.

Scope:

- If *liwork* ≥ 0, *liwork* is **local**.
- If *liwork* = -1, *liwork* is **global**.

Specified as: a fullword integer; where:

- If *liwork* = 0, PDSYGVX and PZHEGVX dynamically allocate the work area used by the subroutine. The work area is deallocated before control is returned to the calling program. This option is an extension to the ScaLAPACK standard.
- If *liwork* = -1, PDSYGVX and PZHEGVX performs a work area query and returns the minimum required size of *iwork* in *iwork*<sub>1</sub>. No computation is performed and the subroutine returns after error checking is complete.
- Otherwise, it must have the following value:

$$liwork \geq \max(isizestein, isizestebz) + 2n$$

where:

- *isizestein* must have the following value:
  - If *jobz* = 'N', *isizestein* = 3*n*+*nprocs*+1
  - If *jobz* = 'V', *isizestein* = (*n*+*jz*-1) + 2*n*+*nprocs*+1
- *isizestebz* = max(4*n*, 14, *nprocs*)
- *nprocs* = (*nprocw*)(*npcol*)

*ifail* See On Return.

*iclustr* See On Return.

*gap* See On Return.

*info* See On Return.

## On Return

*a* *a* is the local part of the global matrix *A*, where:

If *uplo* = 'U', the upper triangle and diagonal of submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  are overwritten; that is, the original input is not preserved.

If *uplo* = 'L', the lower triangle and diagonal of submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  are overwritten; that is, the original input is not preserved.

Scope: **local**

Returned as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 117 on page 698.

- b* is the local part of the global real symmetric or complex Hermitian matrix *B*, containing the results of the Cholesky factorization.
- Scope: **local**
- Specified as: an LLD\_B by (at least) LOCq(N\_B) array, containing numbers of the data type indicated in Table 117 on page 698. Details about the square block-cyclic data distribution of global matrix *B* are stored in *desc\_b*.
- m* is the number of eigenvalues found.
- Scope: **global**
- Returned as: a fullword integer;  $0 \leq m \leq n$ .
- nz* has the following meaning:
- If *jobz*  $\neq$  'V', then *nz* is ignored.
- If *jobz* = 'V', then *nz* is the number of eigenvectors computed—that is, the number of columns of *Z* used in the computation. On output, *nz* = *m* unless you provide insufficient space. To get all the eigenvectors requested, you must supply both sufficient space to hold the eigenvectors in *Z* and sufficient workspace to compute them (see *lwork* and *lrwork*).
- If *range* = 'A' or 'T', this subroutine does not perform any computations if the work space supplied is insufficient. In this case, an input-argument error is issued and your job is terminated. For *range* = 'V', the number of requested eigenvectors is unknown until the eigenvalues are found. In this case, the subroutine computes as many eigenvectors as space allows. Then, if *nz*  $\neq$  *m*, a computational error message is issued.
- Scope: **global**
- Returned as: a fullword integer;  $0 \leq nz \leq m$ .
- w* On normal exit (see *info*), it is the vector *w*, containing the selected eigenvalues in ascending order in the first *m* elements of *w*.
- Scope: **global**
- Returned as: a one-dimensional array of (at least) length *n*, containing numbers of the data type indicated in Table 117 on page 698.
- z* has the following meaning:
- If *jobz* = 'N', then *z* is ignored.
- If *jobz* = 'V' and there is a normal exit (see *info*), then this is the updated local part of the global matrix *Z*, where columns *jz* to *jz+m-1* of the global matrix *Z* contain the orthonormal eigenvectors, corresponding to the selected eigenvalues. If an eigenvector fails to converge, then the corresponding column of the global matrix *Z* contains the last approximation to the eigenvector, and the index of the eigenvector is returned in *ifail*.
- This identifies the **first element** of the local array *Z*. This subroutine computes the location of the first element of the local subarray used, based on *iz*, *jz*, *desc\_z*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading LOCp(*iz+n-1*) by LOCq(*jz+n-1*) part of the local array *Z* must contain the local pieces of the leading *iz+n-1* by *jz+n-1* part of the global matrix *Z*.
- Scope: **local**
- Returned as: an LLD\_Z by (at least) LOCq(N\_Z) array, containing numbers of the data type indicated in Table 117 on page 698.

*work* is the work area used by this subroutine if  $lwork \neq 0$ , where:

If  $lwork \neq 0$  and  $lwork \neq -1$ , its size is (at least) of length  $lwork$ .

If  $lwork = -1$ , its size is (at least) of length 1.

Scope: **local**

Returned as: an area of storage, containing numbers of the data type indicated in Table 117 on page 698, where:

If  $lwork \geq 1$  or  $lwork = -1$ , then:

- If  $jobz = 'N'$ , then  $work_1$  is set to the minimum  $lwork$  needed.
- If  $jobz = 'V'$ , then:

For PDSYGVX,  $work_1$  is set to the minimum  $lwork$  needed to compute all eigenvectors, but not necessarily sufficient to guarantee orthogonality of the eigenvectors.

For PZHEGVX,  $work_1$  is set to the minimum  $lwork$  needed to compute all eigenvectors.

- Except for  $work_1$ , the contents of  $work$  are overwritten on return.

*rwork* is the work area used by this subroutine if  $lrwork \neq 0$ , where:

If  $lrwork \neq 0$  and  $lrwork \neq -1$ , its size is (at least) of length  $lrwork$ .

If  $lrwork = -1$ , its size is (at least) of length 1.

Scope: **local**

Returned as: an area of storage, containing numbers of the data type indicated in Table 117 on page 698, where:

If  $lrwork \geq 1$  or  $lrwork = -1$ , then:

- If  $jobz = 'N'$ , then  $rwork_1$  is set to the minimum  $lrwork$  value needed.
- If  $jobz = 'V'$ , then  $rwork_1$  is set to the minimum  $lrwork$  value needed to compute all eigenvectors, but not necessarily sufficient to guarantee orthogonality of the eigenvectors.
- Except for  $rwork_1$ , the contents of  $rwork$  are overwritten on return.

*iwork* is the work area used by this subroutine if  $liwork \neq 0$ , where:

If  $liwork \neq 0$  and  $liwork \neq -1$ , then its size is (at least) of length  $liwork$ .

If  $liwork = -1$ , then its size is (at least) of length 1.

Scope: **local**

Returned as: an area of storage, where:

If  $liwork \geq 1$  or  $liwork = -1$ , then  $iwork_1$  is set to the minimum  $liwork$  value and contains numbers of the data type indicated in Table 117 on page 698. Except for  $iwork_1$ , the contents of  $iwork$  are overwritten on return.

*ifail* has the following meaning:

If  $B$  is not positive definite (see *info*), then  $ifail_1$  is set to the order of the smallest minor which is not positive definite.

If  $B$  is positive definite and if  $jobz = 'V'$ , it is vector *ifail*, where:

- If there is a normal exit (see *info*), the first  $m$  elements of *ifail* are zero.
- If there is an error exit (where one or more eigenvectors failed to converge—see *info*), *ifail* contains the indices of the eigenvectors that failed to converge.



Scope: **global**

Returned as: a one-dimensional array of (at least) length  $n$ , containing fullword integers;  $0 \leq ifail_i \leq n$ .

*iclustr* has the following meaning:

If *jobz* = 'N', then *iclustr* is ignored.

If *jobz* = 'V', it is vector *iclustr*, containing the indices of the eigenvectors corresponding to a cluster of eigenvalues that could not be reorthogonalized due to insufficient workspace. Eigenvectors corresponding to clusters of eigenvalues indexed  $iclustr_{2i-1}$  to  $iclustr_{2i}$  could not be reorthogonalized due to lack of workspace. **Hence, the eigenvectors corresponding to these clusters may not be orthogonal.**

*iclustr* is a zero-terminated vector; that is, the last element of *iclustr* is set to zero. Assuming that  $k$  is the number of clusters, then:

- $iclustr_{2k} \neq 0$  and  $iclustr_{2k+1} = 0$

Scope: **global**

Returned as: a one-dimensional array of (at least) length  $2(nprow)(npcol)$ , containing fullword integers;  $0 \leq iclustr_i \leq n$ .

*gap* has the following meaning:

If *jobz* = 'N', then *gap* is ignored.

If *jobz* = 'V', it is vector *gap*, containing the gap between the eigenvalues whose eigenvectors could not be reorthogonalized. The values in this vector correspond to the clusters indicated by *iclustr*. As a result, the dot product between the eigenvectors corresponding to the  $i$ -th cluster may be as high as  $(C)(n)/gap_i$ , where  $C$  is a small constant.

Scope: **global**

Returned as: a one-dimensional array of (at least) length  $(nprow)(npcol)$ , containing numbers of the data type indicated in Table 117 on page 698.

*info* has the following meaning:

If *info* = 0, then no input-argument errors or computational errors occurred. This indicates a normal exit.

**Note:** One use of *info* in ScaLAPACK is to identify whether input-argument errors occurred. Because Parallel ESSL terminates the application if input-argument errors occur, the setting of *info* is irrelevant for these errors.

If *info* > 0, then one or more of the following computational errors occurred and the appropriate error messages were issued, indicating an error exit, where:

- If  $\text{mod}(info, 2) \neq 0$ , then one or more eigenvectors failed to converge. Their indices are stored in *ifail*. (Ensure that  $abstol = (2)(unfl)$ .)
- If  $\text{mod}(info/2, 2) \neq 0$ , then the eigenvectors corresponding to one or more clusters of eigenvalues could not be reorthogonalized because of insufficient workspace. The indices of the clusters are stored in *iclustr*.
- If  $\text{mod}(info/4, 2) \neq 0$ , then all the eigenvectors between  $vl$  and  $vu$  could not be computed because of insufficient space. The number of eigenvectors computed is returned in *nz*.



- If  $\text{mod}(\text{info}/8, 2) \neq 0$ , then one of more eigenvalues were not computed. (Ensure that  $\text{abstol} = (2)(\text{unfl})$ .)
- If  $\text{mod}(\text{info}/16, 2) \neq 0$ , then  $B$  was not positive definite.  $\text{ifail}_1$  indicates the order of the smallest minor which is not positive definite.

Scope: **global**

Returned as: a fullword integer;  $\text{info} \geq 0$ .

## Notes and Coding Rules

1. This subroutine accepts lowercase letters for the *jobz*, *range*, and *uplo* arguments.
2. In your C program, argument *info* must be passed by reference.
3. *A*, *B*, *Z*, *w*, *ifail*, *iclustr*, *gap*, *work*, *rwork*, and *iwork* must have no common elements; otherwise, results are unpredictable.
4. The NUMROC utility subroutine can be used to determine the values of LOCp(M\_) and LOCq(N\_) used in the argument descriptions above. For details, see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28 and “NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process” on page 868.
5. The global matrix *A* must be distributed using a square block-cyclic distribution; that is,  $\text{MB\_A} = \text{NB\_A}$ .
6. The global matrix *A* must be aligned on a block boundary; that is:
  - $\text{ia}-1$  must be a multiple of  $\text{MB\_A}$
  - $\text{ja}-1$  must be a multiple of  $\text{NB\_A}$
7. In the process grid, the process row containing the first row of the submatrix *A* must also contain the first row of the submatrix *B*; that is:  $\text{iarow} = \text{ibrow}$  where:
  - $\text{iarow} = \text{mod}(\text{RSRC\_A} + (\text{ia}-1)/\text{MB\_A}, p)$
  - $\text{ibrow} = \text{mod}(\text{RSRC\_B} + (\text{ib}-1)/\text{MB\_B}, p)$
8. In the process grid, the process column containing the first column of the submatrix *A* must also contain the first column of the submatrix *B*; that is:  $\text{iacol} = \text{ibcol}$  where:
  - $\text{iacol} = \text{mod}(\text{CSRC\_A} + (\text{ja}-1)/\text{NB\_A}, q)$
  - $\text{ibcol} = \text{mod}(\text{CSRC\_B} + (\text{jb}-1)/\text{NB\_B}, q)$
9. The block row offset of the global matrix *A* must be equal to the block row offset of the global matrix *B*; that is:
  - $\text{mod}((\text{ia}-1, \text{MB\_A}) = \text{mod}(\text{ib}-1, \text{MB\_B}))$
10. The block column offset of the global matrix *A* must be equal to the block column offset of the global matrix *B*; that is:
  - $\text{mod}((\text{ja}-1, \text{NB\_A}) = \text{mod}(\text{jb}-1, \text{NB\_B}))$
11. The following values must be equal:
  - $\text{MB\_A} = \text{MB\_B}$
  - $\text{NB\_A} = \text{NB\_B}$
  - $\text{CTXT\_A} = \text{CTXT\_B}$
12. If *jobz* = 'V', then also follow these rules:

- In the process grid, the process row containing the first row of the submatrix  $A$  must also contain the first row of the submatrix  $Z$ ; that is:  
 $iarow = izrow$   
 where:
    - $iarow = \text{mod}(\text{RSRC\_A} + (ia-1)/\text{MB\_A}, p)$
    - $izrow = \text{mod}(\text{RSRC\_Z} + (iz-1)/\text{MB\_Z}, p)$
  - $M\_A = M\_Z$
  - $MB\_A = MB\_Z$
  - $NB\_A = NB\_Z$
  - $\text{RSRC\_A} = \text{RSRC\_Z}$
  - $\text{CSRC\_A} = \text{CSRC\_Z}$
  - $\text{CTXT\_A} = \text{CTXT\_Z}$
  - The block row offset of the global matrix  $A$  must be equal to the block row offset of the global general matrix  $Z$ ; that is:  
 –  $\text{mod}((ia-1), \text{MB\_A}) = \text{mod}(iz-1, \text{MB\_Z})$
13. Eigenvectors associated with tightly clustered eigenvalues may not be orthogonal.
  14. Eigenvectors that are on different processes are not reorthogonalized. For details, see the *lwork* and *lrwork* argument.
  15. If *lwork* = -1 on any process, it must equal -1 on all processes. That is, if a subset of the processes specifies -1 for the work area size, they must all specify -1.
  16. If *lrwork* = -1 on any process, it must equal -1 on all processes. That is, if a subset of the processes specifies -1 for the work area size, they must all specify -1.
  17. If *liwork* = -1 on any process, it must equal -1 on all processes. That is, if a subset of the processes specifies -1 for the work area size, they must all specify -1.
  18. The following describes the relationship between workspace, orthogonality, and performance.  
 Let *clustersize* be the number of eigenvalues in the largest cluster, where a cluster is defined as a set of close eigenvalues:
    - $\{w_{k_1}, \dots, w_{k+\text{clustersize}-1} \mid w_{j+1} \leq w_j + \text{orfac}(2)(\text{norm}(A))\}$
    - If *clustersize* is:

$$\text{clustersize} \geq n / \sqrt{(nprow)(npcol)}$$

then providing enough space to compute all the eigenvectors orthogonally causes serious degradation in performance. In the limit (*clustersize* =  $n-1$ ), performance may be no better than using one process.

- If *clustersize* is:

$$\text{clustersize} = n / \sqrt{(nprow)(npcol)}$$

then reorthogonalizing all eigenvectors increases the total execution time by a factor of 2 or more.

- If *clustersize* is:

$$clustersize > n / \sqrt{(nprow)(npcol)}$$

then execution time grows as the square of the cluster size, assuming all other factors remain equal and there is enough workspace. Less workspace means less reorthogonalization, but faster execution.

For PDSYGVX, see the description of *work* on page 704. For PZHEGVX, see the description of *rwork* on page 705.

## Function

This subroutine computes selected eigenvalues and, optionally, the eigenvectors of a real symmetric or complex Hermitian positive definite generalized eigenproblem:

- If *ibtype* = 1, the problem is  $Ax = \lambda Bx$
- If *ibtype* = 2, the problem is  $ABx = \lambda x$
- If *ibtype* = 3, the problem is  $BAx = \lambda x$

In the formulas above:

- *A* represents the global submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$
- *B* represents the global submatrix  $B_{ib:ib+n-1, jb:jb+n-1}$

Eigenvalues and eigenvectors can be selected by specifying a range of values or a range of indices for the eigenvalues. The computation involves the following steps:

1. Compute the Cholesky factorization of *B* using PDPOTRF or PZPOTRF.
2. Reduce the real symmetric or complex Hermitian positive definite generalized eigenproblem to standard form using PDSYGST or PZHEGST.
3. Compute the requested eigenvalues and, optionally, the eigenvectors of the standard form using PDSYEVX or PZHEEVX.
4. Backtransform the eigenvectors to obtain the eigenvectors of the original problem.

## Error Conditions

### Computational Errors

**Note:** For more details, see output argument *info*.

1. The matrix *B* is not positive definite. The order of the smallest minor which is not positive definite is stored in *ifail*<sub>1</sub>.
2. Bisection failed to converge for some eigenvalues. The eigenvalues may not be as accurate as the absolute and relative tolerances.
3. The number of eigenvalues computed does not match the number of eigenvalues requested.
4. No eigenvalues were computed, because the Gershgorin interval initially used was incorrect.
5. Some eigenvectors failed to converge. The indices are stored in *ifail*.
6. Eigenvectors corresponding to one or more clusters of eigenvalues could not be reorthogonalized because of insufficient workspace. The indices of the clusters are stored in *iclustr*.
7. All the eigenvectors between *vl* and *vu* could not be computed due to insufficient workspace. The number of eigenvectors computed is returned in *nz*.

## Resource Errors

1. ( $lwork = 0$ ,  $lrwork = 0$ , or  $liwork = 0$ ) and unable to allocate work space

## Input-Argument and Miscellaneous Errors

### Stage 1:

1. DTYPE\_A is invalid.
2. DTYPE\_B is invalid.
3. DTYPE\_Z is invalid and  $jobz = 'V'$

### Stage 2:

1. CTEXT\_A is invalid.

### Stage 3:

1. This subroutine has been called from outside the process grid.

### Stage 4:

1.  $ibtype \neq 1, 2, \text{ or } 3$
2.  $jobz \neq 'N' \text{ or } 'V'$
3.  $range \neq 'A', 'V', \text{ or } 'T'$
4.  $uplo \neq 'U' \text{ or } 'L'$
5.  $n < 0$
6.  $M\_A < 0$  and  $n = 0$ ;  $M\_A < 1$  otherwise
7.  $N\_A < 0$  and  $n = 0$ ;  $N\_A < 1$  otherwise
8.  $MB\_A < 1$
9.  $NB\_A < 1$
10.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
11.  $CSRC\_A < 0$  or  $CSRC\_A \geq q$
12.  $ia < 1$
13.  $ja < 1$
14.  $M\_B < 0$  and  $n = 0$ ;  $M\_B < 1$  otherwise
15.  $N\_B < 0$  and  $n = 0$ ;  $N\_B < 1$  otherwise
16.  $MB\_B < 1$
17.  $NB\_B < 1$
18.  $RSRC\_B < 0$  or  $RSRC\_B \geq p$
19.  $CSRC\_B < 0$  or  $CSRC\_B \geq q$
20.  $ib < 1$
21.  $jb < 1$
22.  $CTXT\_A \neq CTXT\_B$   
If  $jobz = 'V'$ :
23.  $M\_Z < 0$  and  $n = 0$ ;  $M\_Z < 1$  otherwise
24.  $N\_Z < 0$  and  $n = 0$ ;  $N\_Z < 1$  otherwise
25.  $MB\_Z < 1$
26.  $NB\_Z < 1$
27.  $RSRC\_Z < 0$  or  $RSRC\_Z \geq p$
28.  $CSRC\_Z < 0$  or  $CSRC\_Z \geq q$
29.  $iz < 1$
30.  $jz < 1$
31.  $CTXT\_A \neq CTXT\_Z$

### Stage 5:

1.  $vu \leq vl$  and  $range = 'V'$  and  $n \neq 0$
2.  $il < 1$  and  $range = 'T'$
3.  $(iu < \min(n, il) \text{ or } iu > n)$  and  $range = 'T'$   
If  $n \neq 0$ :
4.  $ia > M\_A$

5.  $ja > N\_A$
6.  $ia+n-1 > M\_A$
7.  $ja+n-1 > N\_A$
8.  $ib > M\_B$
9.  $jb > N\_B$
10.  $ib+n-1 > M\_B$
11.  $jb+n-1 > N\_B$ 
  - If  $n \neq 0$  and  $jobz = 'V'$ :
12.  $iz > M\_Z$
13.  $jz > N\_Z$
14.  $iz+n-1 > M\_Z$
15.  $jz+n-1 > N\_Z$ 
  - In all cases:
16.  $MB\_A \neq NB\_A$
17.  $\text{mod}(ia-1, MB\_A) \neq 0$
18.  $\text{mod}(ja-1, NB\_A) \neq 0$
19.  $MB\_A \neq MB\_B$
20.  $NB\_A \neq NB\_B$
21.  $\text{mod}(ib-1, MB\_B) \neq \text{mod}(ia-1, MB\_A)$
22.  $\text{mod}(jb-1, NB\_B) \neq \text{mod}(ja-1, NB\_A)$
23. In the process grid, the process row containing the first row of the submatrix  $A$  does not contain the first row of the submatrix  $B$ ; that is,  $iarow \neq ibrow$ , where:
  - $iarow = \text{mod}(\text{RSRC\_A} + (ia-1)/MB\_A, p)$
  - $ibrow = \text{mod}(\text{RSRC\_B} + (ib-1)/MB\_B, p)$
24. In the process grid, the process column containing the first column of the submatrix  $A$  does not contain the first column of the submatrix  $B$ ; that is,  $iacol \neq ibcol$ , where:
  - $iacol = \text{mod}(\text{CSRC\_A} + (ja-1)/NB\_A, q)$
  - $ibcol = \text{mod}(\text{CSRC\_B} + (jb-1)/NB\_B, q)$
  - If  $jobz = 'V'$ :
25.  $M\_A \neq M\_Z$
26.  $MB\_A \neq MB\_Z$
27.  $NB\_A \neq NB\_Z$
28.  $\text{mod}(iz-1, MB\_Z) \neq \text{mod}(ia-1, MB\_A)$
29. In the process grid, the process row containing the first row of the submatrix  $A$  does not contain the first row of the submatrix  $Z$ ; that is,  $iarow \neq izrow$ , where:
  - $iarow = \text{mod}(\text{RSRC\_A} + (ia-1)/MB\_A, p)$
  - $izrow = \text{mod}(\text{RSRC\_Z} + (iz-1)/MB\_Z, p)$
30.  $\text{RSRC\_A} \neq \text{RSRC\_Z}$
31.  $\text{CSRC\_A} \neq \text{CSRC\_Z}$

**Stage 6:**

1.  $LLD\_A < \max(1, \text{LOCp}(M\_A))$
2.  $LLD\_B < \max(1, \text{LOCp}(M\_B))$
3.  $lwork \neq 0$ ,  $lwork \neq -1$ , and  $lwork < (\text{minimum value})$ . (For the minimum value, see the  $lwork$  argument description.)
4.  $lrwork \neq 0$ ,  $lrwork \neq -1$ , and  $lrwork < (\text{minimum value})$ . (For the minimum value, see the  $lrwork$  argument description.)
5.  $liwork \neq 0$ ,  $liwork \neq -1$ , and  $liwork < (\text{minimum value})$ . (For the minimum value, see the  $liwork$  argument description.)
- If  $jobz = 'V'$ :
6.  $LLD\_Z < \max(1, \text{LOCp}(M\_Z))$

## Stage 7:

Each of the following global input arguments are checked to determine whether its value differs from the value specified on process  $P_{00}$ :

1. *ibtype* differs.
2. *jobz* differs.
3. *range* differs.
4. *uplo* differs.
5. *n* differs.
6. *ia* differs.
7. *ja* differs.
8. *DTYPE\_A* differs.
9. *M\_A* differs.
10. *N\_A* differs.
11. *MB\_A* differs.
12. *NB\_A* differs.
13. *RSRC\_A* differs.
14. *CSRC\_A* differs.
15. *ib* differs.
16. *jb* differs.
17. *DTYPE\_B* differs.
18. *M\_B* differs.
19. *N\_B* differs.
20. *MB\_B* differs.
21. *NB\_B* differs.
22. *RSRC\_B* differs.
23. *CSRC\_B* differs.
24. *ABSTOL* differs.

Also:

25. *lwork* = -1 on a subset of processes.
26. *lrwork* = -1 on a subset of processes.
27. *liwork* = -1 on a subset of processes.

## Stage 8:

Each of the following global input arguments are checked to determine whether its value differs from the value specified on process  $P_{00}$ :

If *range* = 'V':

1. *vl* differs.
2. *vu* differs.

If *range* = 'I':

3. *il* differs.
4. *iu* differs.

If *jobz* = 'V':

5. *iz* differs.
6. *jz* differs.
7. *DTYPE\_Z* differs.
8. *M\_Z* differs.
9. *N\_Z* differs.
10. *MB\_Z* differs.
11. *NB\_Z* differs.
12. *RSRC\_Z* differs.
13. *CSRC\_Z* differs.
14. *ORFAC* differs.

## Examples

### Example 1

This example shows how to find all the eigenvalues and eigenvectors of a real symmetric positive definite generalized eigenproblem using a  $2 \times 2$  process grid.

#### Notes:

1. Because *range* = 'A', arguments *vl*, *vu*, *il*, and *iu* are not referenced.
2. Because *lwork* = 0 and *liwork* = 0, PDSYGVX dynamically allocates the work areas used by this subroutine.

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONXTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      IBTYPE JOBZ RANGE UPLO  N  A  IA  JA  DESC_A  B  IB  JB  DESC_B  VL  VU  IL  IU  ABSTOL
CALL PDSYGVX( 1, 'V', 'A', 'L', 4, A, 1, 1, DESC_A, B, 1, 1, DESC_B, 0.0D0, 0.0D0, 0, 0, -1.0,

+
      M  NZ  W  ORFAC  Z  IZ  JZ  DESC_Z  WORK  LWORK  IWORK  LIWORK  IFAIL  ICLUSTER  GAP  INFO
      M, NZ, W, -1.0D0, Z, 1, 1, DESC_Z, WORK, 0, IWORK, 0, IFAIL, ICLUSTER, GAP, INFO)
```

	DESC_A	DESC_B	DESC_Z
DTYPE_	1	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	4	4	4
N_	4	4	4
MB_	1	1	1
NB_	1	1	1
RSRC_	0	0	0
CSRC_	0	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>	See below <sup>2</sup>

#### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:

```
LLD_A = MAX(1, NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_B = MAX(1, NUMROC(M_B, MB_B, MYROW, RSRC_B, NPROW))
LLD_Z = MAX(1, NUMROC(M_Z, MB_Z, MYROW, RSRC_Z, NPROW))
```

In this example,  $LLD_A = LLD_B = LLD_Z = 2$  on all processes.

Global real symmetric matrix *A* of order 4, stored in lower storage mode, with block sizes  $1 \times 1$ :

```
B,D    0      1      2      3
0  [ -1.0 | . | . | . ]
1  [ 1.0  | 1.0 | . | . ]
```

## PDSYGVX and PZHEGVX

$$\begin{array}{c} 2 \\ 3 \end{array} \left[ \begin{array}{c|c|c|c} -1.0 & -1.0 & 1.0 & . \\ \hline 1.0 & 1.0 & -1.0 & 1.0 \end{array} \right]$$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$
3		

Local arrays for  $A$ :

p,q	0	1
0	$\begin{array}{cc} -1.0 & . \\ -1.0 & 1.0 \end{array}$	$\begin{array}{cc} . & . \\ -1.0 & . \end{array}$
1	$\begin{array}{cc} 1.0 & . \\ 1.0 & -1.0 \end{array}$	$\begin{array}{cc} 1.0 & . \\ 1.0 & 1.0 \end{array}$

Global real symmetric positive definite matrix  $B$  of order 4, stored in lower storage mode, with block sizes  $1 \times 1$ :

B,D	0	1	2	3
0	$\begin{array}{c} 2.0 \\ \hline 1.0 \end{array}$	$\begin{array}{c} . \\ \hline 2.0 \end{array}$	$\begin{array}{c} . \\ \hline 2.0 \end{array}$	$\begin{array}{c} . \\ \hline . \end{array}$
1				
2	$\begin{array}{c} 0.0 \\ \hline 0.0 \end{array}$	$\begin{array}{c} 1.0 \\ \hline 0.0 \end{array}$	$\begin{array}{c} 2.0 \\ \hline 1.0 \end{array}$	$\begin{array}{c} . \\ \hline 2.0 \end{array}$
3				

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$
3		

Local arrays for  $B$ :

p,q	0	1
0	$\begin{array}{cc} 2.0 & . \\ 0.0 & 2.0 \end{array}$	$\begin{array}{cc} . & . \\ 1.0 & . \end{array}$
1	$\begin{array}{cc} 1.0 & . \\ 0.0 & 1.0 \end{array}$	$\begin{array}{cc} 2.0 & . \\ 0.0 & 2.0 \end{array}$

### Output:

The lower triangle, including the diagonal, of the global real symmetric matrix  $A$  is overwritten; i.e., the original input is not preserved.

On all processes,  $m = 4$  and  $nz = 4$ .



Global vector  $w$  of length 4 is the same on all processes:

- $w = (-1.5526, 0.0000, 0.0000, 5.1526)$

Global real symmetric positive definite matrix  $B$  of order 4, stored in lower storage mode, with block sizes  $1 \times 1$ :

B,D	0	1	2	3
0	1.4142	.	.	.
1	0.7071	1.2247	.	.
2	0.0000	0.8165	1.1547	.
3	0.0000	0.0000	0.8660	1.1180

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$
3		

Local arrays for  $B$ :

p,q	0	1
0	1.4142 0.0000	. 1.1547
1	0.7071 0.0000	. 0.8660

Global general matrix  $Z$  of order 4, with block sizes  $1 \times 1$ :

B,D	0	1	2	3
0	0.8842	0.0000	0.0000	0.1349
1	-0.5619	0.3634	-0.4098	-0.7643
2	0.3072	0.4266	0.1343	0.9517
3	-0.1198	0.0631	0.5441	-0.6969

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$
3		

Local arrays for  $Z$ :

p,q	0	1
0	0.8842 0.3072	0.0000 0.1343
1	0.0000 0.4266	0.1349 0.9517

-----	-----	-----
1	-0.5619   -0.4098	0.3634   -0.7643
	-0.1198   0.5441	0.0631   -0.6969

Global vector *ifail* of length 4 is the same on all processes:

- *ifail* = (0, 0, 0, 0)

Global vector *iclustr* of length 8 ( = 2(*npro*w)(*npcol*)) is the same on all processes:

- *iclustr* = (0, 0, 0, 0, 0, 0, 0, 0)

Global vector *gap* of length 4 ( = (*npro*w)(*npcol*)) is the same on all processes:

- *gap* = (-1.0, -1.0, -1.0, -1.0)

The value of *info* is 0 on all processes.

## Example 2

This example shows how to find all the eigenvalues and eigenvectors of a complex Hermitian positive definite generalized eigenproblem using a 2 × 2 process grid.

### Notes:

1. Because *range* = 'A', arguments *vl*, *vu*, *il*, and *iu* are not referenced.
2. Because *lwork* = 0, *lrwork* = 0, and *liwork* = 0, PZHEGVX dynamically allocates the work areas used by this subroutine.

### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      IBTYPE JOBZ RANGE UPLO  N  A  IA  JA  DESC_A  B  IB  JB  DESC_B  VL      VU  IL  IU  ABSTOL
CALL PZHEGVX( 1, 'V', 'A', 'L', 4, A, 1, 1, DESC_A, B, 1, 1, DESC_B, 0.0D0, 0.0D0, 0, 0, -1.0,

+      M  NZ  W  ORFAC  Z  IZ  JZ  DESC_Z  WORK  LWORK  RWORK  LRWORK  IWORK  LIWORK  IFAIL  ICLUSTER
      M, NZ, W, -1.0D0, Z, 1, 1, DESC_Z, WORK, 0, RWORK, 0, IWORK, 0, IFAIL, ICLUSTER,

+      GAP  INFO
      GAP, INFO)
```

	DESC_A	DESC_B	DESC_Z
DTYPE_	1	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	4	4	4
N_	4	4	4
MB_	1	1	1
NB_	1	1	1
RSRC_	0	0	0
CSRC_	0	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>	See below <sup>2</sup>

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.

2. Each process should set the LLD\_ as follows:

LLD\_A = MAX(1, NUMROC(M\_A, MB\_A, MYROW, RSRC\_A, NPROW))

LLD\_B = MAX(1, NUMROC(M\_B, MB\_B, MYROW, RSRC\_B, NPROW))

LLD\_Z = MAX(1, NUMROC(M\_Z, MB\_Z, MYROW, RSRC\_Z, NPROW))

In this example, LLD\_A = LLD\_B = LLD\_Z = 2 on all processes.

Global complex Hermitian matrix  $A$  of order 4, stored in lower storage mode, with block sizes  $1 \times 1$ :

B,D	0	1	2	3
0	( 1.0, 0.0)	.	.	.
1	( 5.0, -2.0)	( 10.0, 0.0)	.	.
2	( 7.0, 4.0)	( 15.0, 6.0)	( 20.0, 0.0)	.
3	( 9.0, -6.0)	( 20.0, -4.0)	( 25.0, -9.0)	( 30.0, 0.0)

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>
3		

Local arrays for  $A$ :

p,q	0	1
0	( 1.0, . ) ( 7.0, 4.0) ( 20.0, . )	( 15.0, 6.0) .
1	( 5.0, -2.0) . ( 9.0, 6.0) ( 25.0, -9.0)	( 10.0, . ) ( 20.0, -4.0) ( 30.0, . )

Global complex Hermitian positive definite matrix  $B$  of order 4, stored in lower storage mode, with block sizes  $1 \times 1$ :

B,D	0	1	2	3
0	( 9.0, 0.0)	.	.	.
1	( 3.0, -3.0)	( 18.0, 0.0)	.	.
2	( 3.0, 3.0)	( 8.0, 6.0)	( 27.0, 0.0)	.
3	( 3.0, -3.0)	( 8.0, -6.0)	( 12.0, -9.0)	( 61.0, 0.0)

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>
3		

Local arrays for  $B$ :

p,q	0	1
0	$\begin{pmatrix} 9.0, & . \\ 3.0, & 3.0 \end{pmatrix} \quad (27.0, \quad .)$	$(8.0, \quad 6.0) \quad .$
1	$\begin{pmatrix} 3.0, & -3.0 \\ 3.0, & -3.0 \end{pmatrix} \quad (12.0, \quad -9.0)$	$\begin{pmatrix} 18.0, & . \\ 8.0, & -6.0 \end{pmatrix} \quad (61.0, \quad .)$

**Output:**

The lower triangle, including the diagonal, of the global complex Hermitian matrix  $A$  is overwritten; i.e., the original input is not preserved.

On all processes,  $m = 4$  and  $nz = 4$ .

Global vector  $w$  of length 4 is the same on all processes:

- $w = (-0.7969, -0.1152, -0.1669, -1.2304)$

Global complex Hermitian positive definite matrix  $B$  of order 4, stored in lower storage mode, with block sizes  $1 \times 1$ :

B,D	0	1	2	3
0	$(3.0, 0.0)$	.	.	.
1	$(1.0, -1.0)$	$(4.0, 0.0)$	.	.
2	$(1.0, 1.0)$	$(2.0, 1.0)$	$(4.4721, 0.0)$	.
3	$(1.0, -1.0)$	$(1.5, -1.5)$	$(2.3479, -.5590)$	$(6.9767, 0.0)$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$
3		

Local arrays for  $B$ :

p,q	0	1
0	$\begin{pmatrix} 3.0, & 0.0 \\ 1.0, & 1.0 \end{pmatrix} \quad (4.4721, 0.0)$	$(2.0, 1.0) \quad .$
1	$\begin{pmatrix} 1.0, & -1.0 \\ 1.0, & -1.0 \end{pmatrix} \quad (2.3479, -.5590)$	$\begin{pmatrix} 4.0, & 0.0 \\ 1.5, & -1.5 \end{pmatrix} \quad (6.9767, 0.0)$

Global general matrix  $Z$  of order 4, with block sizes  $1 \times 1$ :

B,D	0	1	2	3
0	$(.2784, -.0218)$	$(-.1300, -.0562)$	$(.1842, -.0064)$	$(-.0692, .0118)$
1	$(.0601, .1506)$	$(.1844, .1016)$	$(-.0064, -.0516)$	$(-.0948, .0008)$
2	$(.0000, -.1675)$	$(.0004, -.0082)$	$(-.0591, .1230)$	$(-.0946, .0178)$
3	$(-.0406, .0627)$	$(-.0726, -.0362)$	$(-.0414, -.0635)$	$(-.0513, -.0023)$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$
3		

Local arrays for **Z**:

p,q	0		1	
0	(.2784, -.0218)	(.1842, -.0064)	(-.1300, -.0562)	(-.0692, .0118)
	(.0000, -.1675)	(-.0591, .1230)	(.0004, -.0082)	(-.0946, .0178)
1	(.0601, .1506)	(-.0064, -.0516)	(.1844, .1016)	(-.0948, .0008)
	(-.0406, .0627)	(-.0414, -.0635)	(-.0726, -.0362)	(-.0513, -.0023)

Global vector **ifail** of length 4 is the same on all processes:

- **ifail** = (0, 0, 0, 0)

Global vector **iclustr** of length 8 ( =  $2(nprow)(npcol)$ ) is the same on all processes:

- **iclustr** = (0, 0, 0, 0, 0, 0, 0, 0)

Global vector **gap** of length 4 ( =  $(nprow)(npcol)$ ) is the same on all processes:

- **gap** = (-1.0, -1.0, -1.0, -1.0)

The value of **info** is 0 on all processes.

## PDSYTRD and PZHETRD — Reduce a Real Symmetric or Complex Hermitian Matrix to Tridiagonal Form

### Purpose

PDSYTRD reduces a real symmetric matrix  $A$  to symmetric tridiagonal form  $T$  by an orthogonal similarity transformation:

$$\bullet T = Q^T A Q$$

where  $A$  represents the global real symmetric submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$ .

PZHETRD reduces a complex Hermitian matrix  $A$  to symmetric tridiagonal form  $T$  by a unitary similarity transformation:

$$\bullet T = Q^H A Q$$

where  $A$  represents the global complex Hermitian submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$ .

If  $n = 0$ , no computation is performed and the subroutine returns after doing some parameter checking.

See references [13] and [22].

Table 118. Data Types

$A, \tau, work$	$d, e$	Subroutine
Long-precision real	Long-precision real	PDSYTRD
Long-precision complex	Long-precision real	PZHETRD

### Syntax

<b>Fortran</b>	CALL PDSYTRD   PZHETRD ( <i>uplo</i> , <i>n</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>d</i> , <i>e</i> , <i>tau</i> , <i>work</i> , <i>lwork</i> , <i>info</i> )
<b>C and C++</b>	pdsytrd   pzhetrd ( <i>uplo</i> , <i>n</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>d</i> , <i>e</i> , <i>tau</i> , <i>work</i> , <i>lwork</i> , <i>info</i> );

### On Entry

*uplo* indicates whether the upper or lower triangular part of the global submatrix  $A$  is referenced, where:

If *uplo* = 'U', the upper triangular part is referenced.

If *uplo* = 'L', the lower triangular part is referenced.

Scope: **global**

Specified as: a single character; *uplo* = 'U' or 'L'.

*n* is the order of submatrix  $A$  used in the computation.

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

*a* is the local part of the global real symmetric or complex Hermitian matrix  $A$ . This identifies the **first element** of the local array  $A$ . This subroutine computes the location of the first element of the local subarray used, based on *ia*, *ja*, *desc\_a*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading LOCp(*ia*+*n*-1) by LOCq(*ja*+*n*-1) part of the local array  $A$  must contain the local pieces of the leading *ia*+*n*-1 by *ja*+*n*-1 part of the global matrix, and:

- If  $uplo = 'U'$ , the leading  $n \times n$  upper triangular part of the global submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  must contain the upper triangular part of the submatrix, and the strictly lower triangular part is not referenced.
- If  $uplo = 'L'$ , the leading  $n \times n$  lower triangular part of the global submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  must contain the lower triangular part of the submatrix, and the strictly upper triangular part is not referenced.

Scope: **local**

Specified as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 118 on page 724. Details about the square block-cyclic data distribution of global matrix  $A$  are stored in  $desc\_a$ .

$ia$  is the row index of the global matrix  $A$ , identifying the first row of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq M\_A$  and  $ia+n-1 \leq M\_A$ .

$ja$  is the column index of the global matrix  $A$ , identifying the first column of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$  and  $ja+n-1 \leq N\_A$ .

$desc\_a$  is the array descriptor for global matrix  $A$ , described in the following table:

$desc\_a$	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_A < p$	Global
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < q$	Global
9	LLD_A	The leading dimension of the local array	$LLD\_A \geq \max(1, LOCp(M\_A))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

$d$  See On Return.

$e$  See On Return.

$\tau$  See On Return.

*work* has the following meaning:

If  $lwork = 0$ , *work* is ignored.

If  $lwork \neq 0$ , *work* is the work area used by this subroutine, where:

- If  $lwork \neq -1$ , its size is (at least) of length *lwork*.
- If  $lwork = -1$ , its size is (at least) of length 1.

Scope: **local**

Specified as: an area of storage containing numbers of data type indicated in Table 118 on page 724.

*lwork* is the number of elements in array WORK.

Scope:

- If  $lwork \geq 0$ , *lwork* is **local**
- If  $lwork = -1$ , *lwork* is **global**

Specified as: a fullword integer; where:

- If  $lwork = 0$ , PDSYTRD and PZHETRD dynamically allocate the work area used by the subroutine. The work area is deallocated before control is returned to the calling program. This option is an extension to the ScaLAPACK standard.
- If  $lwork = -1$ , PDSYTRD and PZHETRD perform a work area query and return the minimum size of *work* in  $work_1$ . No computation is performed and the subroutine returns after error checking is complete.
- Otherwise, it must have the following value:

$$lwork \geq \max(nb(np+1), 3nb)$$

where:

- $nb = MB\_A = NB\_A$
- $iarow = \text{mod}(\text{RSRC\_A} + (ia-1)/nb, nprow)$ .
- $np = \text{NUMROC}(n, nb, myrow, iarow, nprow)$

*info* See On Return.

### On Return

*a* is the updated local part of the global matrix *A*, containing the results of the computation, where:

- If *uplo* = 'U', the diagonal and first superdiagonal of  $A_{ia:ia+n-1, ja:ja+n-1}$  are overwritten by the corresponding elements of the tridiagonal matrix *T*. The elements above the first superdiagonal are overwritten with  $v_{1:i-1}$ . These elements with  $\tau$  represent the matrix *Q* as a product of elementary reflectors.
- If *uplo* = 'L', the diagonal and first subdiagonal of  $A_{ia:ia+n-1, ja:ja+n-1}$  are overwritten by the corresponding elements of the tridiagonal matrix *T*. The elements below the first subdiagonal are overwritten with  $v_{i+2:n}$ . These elements with  $\tau$  represent the matrix *Q* as a product of elementary reflectors.

See "Function" on page 728, for more information.

Scope: **local**

Returned as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 118 on page 724. Details about the square block-cyclic data distribution of global matrix *A* are stored in *desc\_a*.



$d$  is the updated local part of the global matrix  $D$ , where  $d_{ja:ja+n-1}$  contains the diagonal elements of the tridiagonal matrix  $T$ .

This identifies the **first element** of the local array  $D$ . This subroutine computes the location of the first element of the local subarray used, based on  $ja$ ,  $desc\_a$ ,  $p$ ,  $q$ ,  $myrow$ , and  $mycol$ ; therefore, the leading 1 by  $LOCq(ja+n-1)$  part of the local array  $D$  must contain the local pieces of the leading 1 by  $ja+n-1$  part of the global matrix  $D$ .

A copy of the vector  $d$ , with a block size of  $NB\_A$  and global index  $ja$ , is returned to each row of the process grid. The process column over which the first column of  $d$  is distributed is  $CSRC\_A$ .

Scope: **local**

Returned as: a 1 by (at least)  $LOCq(N\_A)$  array, containing numbers of the data type indicated in Table 118 on page 724.

$e$  is the updated local part of the global matrix  $E$ , containing the off-diagonal elements of the tridiagonal matrix  $T$ , where:

If  $uplo = 'U'$ , then  $e_{ja} = 0$  and  $e_{ja+1:ja+n-1}$  contains the superdiagonal elements of the tridiagonal matrix  $T$ .

If  $uplo = 'L'$ , then  $e_{ja:ja+n-2}$  contains the subdiagonal elements of the tridiagonal matrix  $T$ , and  $e_{ja+n-1} = 0$ .

This identifies the **first element** of the local array  $E$ . This subroutine computes the location of the first element of the local subarray used, based on  $ja$ ,  $desc\_a$ ,  $p$ ,  $q$ ,  $myrow$ , and  $mycol$ ; therefore, the leading 1 by  $LOCq(ja+n-1)$  part of the local array  $E$  must contain the local pieces of the leading 1 by  $ja+n-1$  part of the global matrix  $E$ .

A copy of the vector  $e$ , with a block size of  $NB\_A$  and global index  $ja$ , is returned to each row of the process grid. The process column over which the first column of  $E$  is distributed is  $CSRC\_A$ .

Scope: **local**

Returned as: a 1 by (at least)  $LOCq(N\_A)$  array, containing numbers of the data type indicated in Table 118 on page 724.

$\tau$  is the updated local part of the global matrix  $\tau$ , containing the scalar factors of the elementary reflectors, where:

If  $uplo = 'U'$ , then  $\tau_{ja}$  is zero and  $\tau_{ja+1:ja+n-1}$  contains the scalar factors of the elementary reflectors.

If  $uplo = 'L'$ , then  $\tau_{ja:ja+n-2}$  contains the scalar factors of the elementary reflectors and  $\tau_{ja+n-1}$  is zero.

This identifies the **first element** of the local array  $\tau$ . This subroutine computes the location of the first element of the local subarray used, based on  $ja$ ,  $desc\_a$ ,  $p$ ,  $q$ ,  $myrow$ , and  $mycol$ ; therefore, the leading 1 by  $LOCq(ja+n-1)$  part of the local array  $\tau$  must contain the local pieces of the leading 1 by  $ja+n-1$  part of the global matrix  $\tau$ .

A copy of the vector  $\tau$ , with a block size of  $NB\_A$  and global index  $ja$ , is returned to each row of the process grid. The process column over which the first column of  $\tau$  is distributed is  $CSRC\_A$ .

Scope: **local**

Returned as: a 1 by (at least)  $LOCq(N\_A)$  array, containing numbers of the data type indicated in Table 118 on page 724.

*work* is the work area used by this subroutine if *lwork*  $\neq$  0, where:

If *lwork*  $\neq$  0 and *lwork*  $\neq$  -1, its size is (at least) of length *lwork*.

If *lwork* = -1, its size is (at least) of length 1.

Scope: **local**

Returned as: an area of storage, where:

If *lwork*  $\geq$  1 or *lwork* = -1, then *work*<sub>1</sub> is set to the minimum *lwork* value and contains numbers of the data type indicated in Table 118 on page 724. Except for *work*<sub>1</sub>, the contents of *work* are overwritten on return.

*info* indicates that a successful computation occurred.

Scope: **global**

Returned as: a fullword integer; *info* = 0.

## Notes and Coding Rules

1. This subroutine accepts lowercase letters for the *uplo* argument.
2. In your C program, argument *info* must be passed by reference.
3. The imaginary parts of the diagonal elements of a complex Hermitian matrix **A** are assumed to be zero, so you do not have to set these values. On output, they are set to zero, except when *n* is equal to zero.
4. Matrix **A**, **d**, **e**,  $\tau$ , and *work* must have no common elements; otherwise, results are unpredictable.
5. The NUMROC utility subroutine can be used to determine the values of LOCp(M\_) and LOCq(N\_) used in the argument descriptions above. For details, see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28 and “NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process” on page 868.
6. The global real symmetric or complex Hermitian matrix **A** must be distributed using a square block-cyclic distribution; that is, MB\_A = NB\_A.
7. The global real symmetric or complex Hermitian matrix **A** must be aligned on a block boundary; that is:
  - *ia*-1 must be a multiple of MB\_A
  - *ja*-1 must be a multiple of NB\_A
8. There are no array descriptors for **d**, **e**, and  $\tau$ . These are all row distributed vectors with block size NB\_A, local arrays of dimension 1 by LOCq(N\_A), and global index *ja*. A copy of these vectors exist on each row of the process grid, and the process column over which the first column of **D**, **E**, and  $\tau$  is distributed is CSRC\_A.
9. For suggested block sizes, see “Coding Tips for Optimizing Parallel Performance” on page 77.
10. If *lwork* = -1 on any process, it must equal -1 on all processes. That is, if a subset of the processes specifies -1 for the work area size, they must all specify -1.

## Function

### PDSYTRD

reduces a real symmetric matrix **A** to symmetric tridiagonal form **T** by an orthogonal similarity transformation:

- $T = Q^T A Q$

where:

- $A$  represents the global real symmetric submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$ .
- Matrix  $Q$  represents the following:
  - For  $uplo = 'U'$ , the matrix  $Q$  is the product of elementary reflectors:

$$Q = H_{n-1} \dots H_2 H_1,$$

where:

- For each  $i$ :  $H_i = I - \tau v v^T$
- $\tau$  is a real scalar
- $v$  is a real vector with  $v_{i+1:n} = 0$  and  $v_i = 1$
- $v_{1:i-1}$  is stored on return in submatrix  $A_{1+(ia-1):i-1+(ia-1), i+1+(ja-1)}$
- $\tau$  is stored on return in  $\tau_{i+(ja-1)}$
- $I$  is the identity matrix

If  $uplo = 'U'$ , then the following example shows the contents of  $A$  on return with  $n = 5$  and  $ia = ja = 1$ :

$$\begin{bmatrix} d & e & v_2 & v_3 & v_4 \\ . & d & e & v_3 & v_4 \\ . & . & d & e & v_4 \\ . & . & . & d & e \\ . & . & . & . & d \end{bmatrix}$$

where:

- $d$  represents the diagonal elements of  $T$
- $e$  represents the superdiagonal elements of  $T$
- $v_i$  represents the corresponding elements of the vector defining  $H_i$ .
- For  $uplo = 'L'$ , the matrix  $Q$  is the product of elementary reflectors:
 
$$Q = H_1 H_2 \dots H_{n-1},$$

where:

- For each  $i$ :  $H_i = I - \tau v v^T$
- $\tau$  is a real scalar
- $v$  is a real vector with  $v_{1:i} = 0$  and  $v_{i+1} = 1$ .
- $v_{i+2:n}$  is stored on return in submatrix  $A_{i+2+(ia-1):n+(ia-1), i+(ja-1)}$ .
- $\tau$  is stored on return in  $\tau_{i+(ja-1)}$
- $I$  is the identity matrix.

If  $uplo = 'L'$ , then the following example shows the contents of  $A$  on return with  $n = 5$  and  $ia = ja = 1$ :

$$\begin{bmatrix} d & . & . & . & . \\ e & d & . & . & . \\ v_1 & e & d & . & . \\ v_1 & v_2 & e & d & . \\ v_1 & v_2 & v_3 & e & d \end{bmatrix}$$

where:

- $d$  represents the diagonal elements of  $T$
- $e$  represents the subdiagonal elements of  $T$
- $v_i$  represents the corresponding elements of the vector defining  $H_i$ .

## PZHETRD

reduces a complex Hermitian matrix  $A$  to symmetric tridiagonal form  $T$  by a unitary similarity transformation:

- $T = Q^H A Q$

where:

- $A$  represents the global complex Hermitian submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$ .
- Matrix  $Q$  represents the following:
  - For  $uplo = 'U'$ , the matrix  $Q$  is the product of elementary reflectors:

$$Q = H_{n-1} \dots H_2 H_1,$$

where:

- For each  $i$ :  $H_i = I - \tau v v^T$
- $\tau$  is a complex scalar
- $v$  is a complex vector with  $v_{i+1:n}$  is (0,0) and  $v_i$  is (1,0)
- $v_{1:i-1}$  is stored on return in submatrix  $A_{1+(ia-1):i-1+(ia-1), i+1+(ja-1)}$
- $\tau$  is stored on return in  $\tau_{i+(ja-1)}$
- $I$  is the identity matrix

If  $uplo = 'U'$ , then the following example shows the contents of  $A$  on return with  $n = 5$  and  $ia = ja = 1$ :

$$\begin{bmatrix} d & e & v_2 & v_3 & v_4 \\ . & d & e & v_3 & v_4 \\ . & . & d & e & v_4 \\ . & . & . & d & e \\ . & . & . & . & d \end{bmatrix}$$

where:

- $d$  represents the diagonal elements of  $T$
- $e$  represents the superdiagonal elements of  $T$
- $v_i$  represents the corresponding elements of the vector defining  $H_i$ .
- For  $uplo = 'L'$ , the matrix  $Q$  is the product of elementary reflectors:
 
$$Q = H_1 H_2 \dots H_{n-1},$$

where:

- For each  $i$ :  $H_i = I - \tau v v^T$
- $\tau$  is a complex scalar
- $v$  is a complex vector with  $v_{1:i}$  is (0,0) and  $v_{i+1}$  is (1,0)
- $v_{i+2:n}$  is stored on return in submatrix  $A_{i+2+(ia-1):n+(ia-1), i+(ja-1)}$
- $\tau$  is stored on return in  $\tau_{i+(ja-1)}$
- $I$  is the identity matrix

If  $uplo = 'L'$ , then the following example shows the contents of  $A$  on return with  $n = 5$  and  $ia = ja = 1$ :

$$\begin{bmatrix} d & . & . & . & . \\ e & d & . & . & . \\ v_1 & e & d & . & . \\ v_1 & v_2 & e & d & . \\ v_1 & v_2 & v_3 & e & d \end{bmatrix}$$

where:

- $d$  represents the diagonal elements of  $T$
- $e$  represents the subdiagonal elements of  $T$
- $v_i$  represents the corresponding elements of the vector defining  $H_i$ .

## Error Conditions

### Computational Errors

None

### Resource Errors

1.  $lwork = 0$  and unable to allocate work space

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1.  $DTYPE\_A$  is invalid.

#### Stage 2:

1.  $CTXT\_A$  is invalid.

#### Stage 3:

1. This subroutine has been called from outside the process grid.

#### Stage 4:

1.  $uplo \neq 'U'$  or  $'L'$
2.  $n < 0$
3.  $M\_A < 0$  and  $n = 0$ ;  $M\_A < 1$  otherwise
4.  $N\_A < 0$  and  $n = 0$ ;  $N\_A < 1$  otherwise
5.  $MB\_A < 1$
6.  $NB\_A < 1$
7.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
8.  $CSRC\_A < 0$  or  $CSRC\_A \geq q$
9.  $ia < 1$
10.  $ja < 1$

#### Stage 5: If $n \neq 0$ :

1.  $ia > M\_A$
2.  $ja > N\_A$
3.  $ia+n-1 > M\_A$
4.  $ja+n-1 > N\_A$

In all cases:

1.  $MB\_A \neq NB\_A$
2.  $\text{mod}(ia-1, MB\_A) \neq 0$
3.  $\text{mod}(ja-1, NB\_A) \neq 0$

#### Stage 6:

1.  $LLD\_A < \max(1, \text{LOCp}(M\_A))$
2.  $lwork \neq 0$ ,  $lwork \neq -1$ , and  $lwork < \max(nb(np+1), 3nb)$

where:

- $nb = MB\_A = NB\_A$
- $iarow = \text{mod}(RSRC\_A+(ia-1)/nb, nprow)$ .
- $np = \text{NUMROC}(n, nb, myrow, iarow, nprow)$

#### Stage 7:

Each of the following global input arguments are checked to determine whether its value differs from the value specified on process  $P_{00}$ :

1.  $uplo$  differs.
2.  $n$  differs.
3.  $ia$  differs.

4. *ja* differs.
5. DTYPE\_A differs.
6. M\_A differs.
7. N\_A differs.
8. MB\_A differs.
9. NB\_A differs.
10. RSRC\_A differs.
11. CSRC\_A differs.
- Also:
12. *lwork* = -1 on a subset of processes.

## Examples

### Example 1

This example shows the reduction of a real symmetric matrix of order 4 to symmetric tridiagonal form, using a  $2 \times 2$  process grid.

**Note:** Because *lwork* = 0, PDSYTRD dynamically allocates the work area used by this subroutine.

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      UPLO  N   A   IA  JA   DESC_A  D   E   TAU   WORK  LWORK  INFO
      |    |   |   |   |         |   |   |   |   |   |   |
CALL PDSYTRD( 'U' , 4 , A   , 1 , 1 , DESC_A , D , E , TAU , WORK , 0   , INFO )
```

	DESC_A
DTYPE_	1
CTXT_	<i>icontxt</i> <sup>1</sup>
M_	4
N_	4
MB_	1
NB_	1
RSRC_	0
CSRC_	0
LLD_	See below <sup>2</sup>

#### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:  
 $LLD\_A = \text{MAX}(1, \text{NUMROC}(M\_A, MB\_A, MYROW, RSRC\_A, NPROW))$   
 In this example,  $LLD\_A = 2$  on all processes.

Global real symmetric matrix *A* of order 4 with block sizes  $1 \times 1$ :

B,D	0	1	2	3
0	5.0	4.0	1.0	1.0
1	.	5.0	1.0	1.0
2	.	.	4.0	2.0
3	.	.	.	4.0

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$
3		

Local arrays for  $A$ :

p,q	0	1
0	5.0 1.0 . 4.0	4.0 1.0 . 2.0
1	. 1.0 . .	5.0 1.0 . 4.0

**Output:**

Global real symmetric matrix  $A$  of order 4 with block sizes  $1 \times 1$ :

B,D	0	1	2	3
0	1.00	0.00	0.41	0.22
1	.	6.00	2.83	0.22
2	.	.	7.00	-2.45
3	.	.	.	4.00

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$
3		

Local arrays for  $A$ :

p,q	0	1
0	1.00 0.41 . 7.00	0.00 0.22 . -2.45
1	. 2.83 . .	6.00 0.22 . 4.00

Global row vector  $D$  of length 4 with block size 1:

## PDSYTRD and PZHETRD

B,D	0	1	2	3
0	[ 1.00   6.00   7.00   4.00 ]			

**Note:** A copy of  $D$  is distributed across each row of the process grid.

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
-----	-----	-----
	P <sub>00</sub>	P <sub>01</sub>
-----	-----	-----
	P <sub>10</sub>	P <sub>11</sub>

Local arrays for  $D$ :

p,q	0	1
-----	-----	-----
0	1.00 7.00	6.00 4.00
-----	-----	-----
1	1.00 7.00	6.00 4.00

Global row vector  $E$  of length 4 with block size 1:

B,D	0	1	2	3
0	[ 0.00   0.00   2.83   -2.45 ]			

**Note:** A copy of  $E$  is distributed across each row of the process grid.

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
-----	-----	-----
	P <sub>00</sub>	P <sub>01</sub>
-----	-----	-----
	P <sub>10</sub>	P <sub>11</sub>

Local arrays for  $E$ :

p,q	0	1
-----	-----	-----
0	0.00 2.83	0.00 -2.45
-----	-----	-----
1	0.00 2.83	0.00 -2.45

Global row vector  $\tau$  of length 4 with block size 1:

B,D	0	1	2	3
0	[ 0.00   0.00   1.71   1.82 ]			

**Note:** A copy of  $\tau$  is distributed across each row of the process grid.

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
-----	-----	-----
	P <sub>00</sub>	P <sub>01</sub>
-----	-----	-----
	P <sub>10</sub>	P <sub>11</sub>

Local arrays for  $\tau$ :



p,q	0		1	
0	0.00	1.71	0.00	1.82
1	0.00	1.71	0.00	1.82

The value of *info* is 0 on all processes.

### Example 2

This example shows the reduction of a complex Hermitian matrix of order 4 to symmetric tridiagonal form, using a  $2 \times 2$  process grid.

#### Note:

1. The imaginary parts of the diagonal elements of a complex Hermitian matrix **A** are assumed to be zero, so you do not have to set these values. On output, they are set to zero, except when *n* is equal to zero.
2. Because *lwork* = 0, PZHETRD dynamically allocates the work area used by this subroutine.

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      UPLO  N   A   IA  JA  DESC_A  D   E   TAU  WORK  LWORK  INFO
      |    |   |   |   |   |   |   |   |   |   |   |
CALL PZHETRD( 'L' , 4 , A , 1 , 1 , DESC_A , D , E , TAU , WORK , 0 , INFO )
```

	DESC_A
DTYPE_	1
CTXT_	<i>icontxt</i> <sup>1</sup>
M_	4
N_	4
MB_	1
NB_	1
RSRC_	0
CSRC_	0
LLD_	See below <sup>2</sup>

#### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:  

$$\text{LLD\_A} = \text{MAX}(1, \text{NUMROC}(\text{M\_A}, \text{MB\_A}, \text{MYROW}, \text{RSRC\_A}, \text{NPROW}))$$
In this example,  $\text{LLD\_A} = 2$  on all processes.

Global complex Hermitian matrix **A** of order 4 with block sizes  $1 \times 1$ :

B,D	0	1	2	3
0	(5.0, 0.0)	.	.	.
1	(4.0, 1.0)	(5.0, 0.0)	.	.

## PDSYTRD and PZHETRD

$$\begin{array}{c|c|c|c|c} 2 & (1.0, 2.0) & (1.0, 0.0) & (4.0, 0.0) & . \\ \hline 3 & (2.0, 3.0) & (3.0, 2.0) & (5.0, 1.0) & (4.0, 0.0) \end{array}$$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>
3		

Local arrays for *A*:

p,q	0	1
0	(5.0, . ) (1.0, 2.0) (4.0, . )	(1.0, 0.0) .
1	(4.0, 1.0) . (2.3, 3.0) (5.0, 1.0)	(5.0, . ) . (3.0, 2.0) (4.0, . )

**Output:**

Global complex Hermitian matrix *A* of order 4 with block sizes  $1 \times 1$ :

B,D	0	1	2	3
0	( 5.00, 0.00)	.	.	.
1	(-5.92, 0.00)	(10.09, 0.00)	.	.
2	( 0.12, 0.19)	( 2.36, 0.00)	( 4.16, 0.0)	.
3	( 0.23, 0.28)	( 0.14, 0.19)	( 1.62, 0.00)	(-1.25, 0.00)

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>
3		

Local arrays for *A*

p,q	0	1
0	( 5.00, 0.00) ( 0.12, 0.19) ( 4.16, 0.00)	( 2.36, 0.00) .
1	(-5.92, 0.00) . ( 0.23, 0.28) ( 1.62, 0.00)	(10.09, 0.00) . ( 0.14, 0.19) (-1.25, 0.00)

Global row vector *D* of length 4 with block size 1:

B,D	0	1	2	3
0	5.00	10.09	4.16	-1.25

**Note:** A copy of *D* is distributed across each row of the process grid.

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
-----	-----	-----
	P <sub>00</sub>	P <sub>01</sub>
-----	-----	-----
	P <sub>10</sub>	P <sub>11</sub>

Local arrays for  $D$ :

p,q	0	1
-----	-----	-----
0	5.00 4.16	10.09 -1.25
-----	-----	-----
1	5.00 4.16	10.09 -1.25

Global row vector  $E$  of length 4 with block size 1:

B,D	0	1	2	3
0	[ -5.92   2.36   1.62   0.00 ]			

**Note:** A copy of  $E$  is distributed across each row of the process grid.

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
-----	-----	-----
	P <sub>00</sub>	P <sub>01</sub>
-----	-----	-----
	P <sub>10</sub>	P <sub>11</sub>

Local arrays for  $E$ :

p,q	0	1
-----	-----	-----
0	-5.92 1.62	2.36 0.00
-----	-----	-----
1	-5.92 1.62	2.36 0.00

Global row vector  $\tau$  of length 4 with block size 1:

B,D	0	1	2	3
0	[ (1.68, 0.17)   (1.87, 0.21)   (1.96, -0.27)   (0.00, 0.00) ]			

**Note:** A copy of  $\tau$  is distributed across each row of the process grid.

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
-----	-----	-----
	P <sub>00</sub>	P <sub>01</sub>
-----	-----	-----
	P <sub>10</sub>	P <sub>11</sub>

Local arrays for  $\tau$ :

p,q	0	1
-----	-----	-----
0	(1.68, 0.17) (1.96, -0.27)	(1.87, 0.21) (0.00, 0.00)
-----	-----	-----
1	(1.68, 0.17) (1.96, -0.27)	(1.87, 0.21) (0.00, 0.00)

## PDSYGST and PZHEGST

The value of *info* is 0 on all processes.

## PDSYGST and PZHEGST — Reduce a Real Symmetric or Complex Hermitian Positive Definite Generalized Eigenproblem to Standard Form

### Purpose

These subroutines reduce a real symmetric or complex Hermitian positive definite generalized eigenproblem to standard form and solves the following problem types:

- If  $ibtype = 1$ , the problem is  $Ax = \lambda Bx$
- If  $ibtype = 2$ , the problem is  $ABx = \lambda x$
- If  $ibtype = 3$ , the problem is  $BAx = \lambda x$

$B$  must have been previously factored by a call to PDPOTRF or PZPOTRF.

In the formulas above:

- $A$  represents the global real symmetric or complex Hermitian submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$
- $B$  represents the global real symmetric or complex Hermitian submatrix  $B_{ib:ib+n-1, jb:jb+n-1}$

If  $n = 0$ , no computation is performed and the subroutine returns after doing some parameter checking.

See reference [13].

Table 119. Data Types

$A, B$	$scale$	Subroutine
Long-precision real	Long-precision real	PDSYGST
Long-precision complex	Long-precision real	PZHEGST

### Syntax

Fortran	CALL PDSYGST PZHEGST ( $ibtype, uplo, n, a, ia, ja, desc\_a, b, ib, jb, desc\_b, scale, info$ )
C and C++	pdsygst pzhegst ( $ibtype, uplo, n, a, ia, ja, desc\_a, b, ib, jb, desc\_b, scale, info$ );

### On Entry

$ibtype$  specifies the problem type, where:

- If  $ibtype = 1$ , the problem is  $Ax = \lambda Bx$
- If  $ibtype = 2$ , the problem is  $ABx = \lambda x$
- If  $ibtype = 3$ , the problem is  $BAx = \lambda x$

Scope: **global**

Specified as: a fullword integer;  $ibtype = 1, 2$ , or  $3$ .

$uplo$  indicates whether the upper or lower triangular part of the global submatrix  $A$  is referenced, and how the global submatrix  $B$  has been factored, where:

If  $uplo = 'U'$ , the upper triangular part is referenced.

If  $uplo = 'L'$ , the lower triangular part is referenced.

Scope: **global**

Specified as: a single character;  $uplo = 'U'$  or  $'L'$ .

$n$  is the order of submatrices  $A$  and  $B$  used in the computation.

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

$a$  is the local part of the global real symmetric or complex Hermitian matrix  $A$ . This identifies the **first element** of the local array  $A$ . This subroutine computes the location of the first element of the local subarray used, based on  $ia$ ,  $ja$ ,  $desc\_a$ ,  $p$ ,  $q$ ,  $myrow$ , and  $mycol$ ; therefore, the leading  $LOCp(ia+n-1)$  by  $LOCq(ja+n-1)$  part of the local array  $A$  must contain the local pieces of the leading  $ia+n-1$  by  $ja+n-1$  part of the global matrix, and:

- If  $uplo = 'U'$ , the leading  $n \times n$  upper triangular part of the global submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  must contain the upper triangular part of the submatrix, and the strictly lower triangular part is not referenced.
- If  $uplo = 'L'$ , the leading  $n \times n$  lower triangular part of the global submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  must contain the lower triangular part of the submatrix, and the strictly upper triangular part is not referenced.

Scope: **local**

Specified as: an  $LLD\_A$  by (at least)  $LOCq(N\_A)$  array, containing numbers of the data type indicated in Table 119 on page 739. Details about the square block-cyclic data distribution of global matrix  $A$  are stored in  $desc\_a$ .

$ia$  is the row index of the global matrix  $A$ , identifying the first row of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq M\_A$  and  $ia+n-1 \leq M\_A$ .

$ja$  is the column index of the global matrix  $A$ , identifying the first column of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$  and  $ja+n-1 \leq N\_A$ .

$desc\_a$  is the array descriptor for global matrix  $A$ , described in the following table:

$desc\_a$	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_A < p$	Global

<i>desc_a</i>	Name	Description	Limits	Scope
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq \text{CSRC\_A} < q$	Global
9	LLD_A	The leading dimension of the local array	$\text{LLD\_A} \geq \max(1, \text{LOCp}(\text{M\_A}))$	Local

Specified as: an array of (at least) length 9, containing fullword integers.

*b* is the local part of the global real symmetric or complex Hermitian matrix *B*, containing the results of the Cholesky factorization computed by PDPOTRF or PZPOTRF. This identifies the **first element** of the local array *B*. This subroutine computes the location of the first element of the local subarray used, based on *ib*, *jb*, *desc\_b*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading  $\text{LOCp}(\text{ib}+n-1)$  by  $\text{LOCq}(\text{jb}+n-1)$  part of the local array *B* must contain the local pieces of the leading  $\text{ib}+n-1$  by  $\text{jb}+n-1$  part of the global matrix.

Scope: **local**

Specified as: an  $\text{LLD\_B}$  by (at least)  $\text{LOCq}(\text{N\_B})$  array, containing numbers of the data type indicated in Table 119 on page 739. Details about the square block-cyclic data distribution of global matrix *B* are stored in *desc\_b*.

*ib* is the row index of the global matrix *B*, identifying the first row of the submatrix *B*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq \text{ib} \leq \text{M\_B}$  and  $\text{ib}+n-1 \leq \text{M\_B}$ .

*jb* is the column index of the global matrix *B*, identifying the first column of the submatrix *B*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq \text{jb} \leq \text{N\_B}$  and  $\text{jb}+n-1 \leq \text{N\_B}$ .

*desc\_b* is the array descriptor for global matrix *B*, described in the following table:

<i>desc_b</i>	Name	Description	Limits	Scope
1	DTYPE_B	Descriptor type	$\text{DTYPE\_B}=1$	Global
2	CTXT_B	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_B	Number of rows in the global matrix	If $n = 0$ : $\text{M\_B} \geq 0$ Otherwise: $\text{M\_B} \geq 1$	Global
4	N_B	Number of columns in the global matrix	If $n = 0$ : $\text{N\_B} \geq 0$ Otherwise: $\text{N\_B} \geq 1$	Global
5	MB_B	Row block size	$\text{MB\_B} \geq 1$	Global
6	NB_B	Column block size	$\text{NB\_B} \geq 1$	Global
7	RSRC_B	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq \text{RSRC\_B} < p$	Global

## PDSYGST and PZHEGST

<i>desc_b</i>	Name	Description	Limits	Scope
8	CSRC_B	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq \text{CSRC\_B} < q$	Global
9	LLD_B	The leading dimension of the local array	$\text{LLD\_B} \geq \max(1, \text{LOCp}(\text{M\_B}))$	Local

Specified as: an array of (at least) length 9, containing fullword integers.

*scale* See On Return.

*info* See On Return.

### On Return

*a* is the updated local part of the global matrix *A*, containing the transformed matrix, where:

- For PDSYGST:
  - If *ibtype* = 1, global matrix *A* is overwritten by:
    - $A = U^{-T}AU^{-1}$  if *uplo* = 'U'.
    - $A = L^{-1}AL^{-T}$  if *uplo* = 'L'.
  - If *ibtype* = 2 or 3, global matrix *A* is overwritten by:
    - $A = UAU^T$  if *uplo* = 'U'.
    - $A = L^TAL$  if *uplo* = 'L'.
- For PZHEGST:
  - If *ibtype* = 1, global matrix *A* is overwritten by:
    - $A = U^{-H}AU^{-1}$  if *uplo* = 'U'.
    - $A = L^{-1}AL^{-H}$  if *uplo* = 'L'.
  - If *ibtype* = 2 or 3, global matrix *A* is overwritten by:
    - $A = UAU^H$  if *uplo* = 'U'.
    - $A = L^H AL$  if *uplo* = 'L'.

See “Function” on page 743, for more information.

Scope: **local**

Returned as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 119 on page 739. Details about the square block-cyclic data distribution of global matrix *A* are stored in *desc\_a*.

*scale* reserved for future use.

Scope: **global**

Returned as: a long-precision real number; *scale* = 1.0

*info* indicates that a successful computation occurred.

Scope: **global**

Returned as: a fullword integer; *info* = 0.

## Notes and Coding Rules

1. This subroutine accepts lowercase letters for the *uplo* argument.
2. In your C program, arguments *scale* and *info* must be passed by reference.
3. Matrices *A* and *B* must have no common elements; otherwise, results are unpredictable.



4. The NUMROC utility subroutine can be used to determine the values of  $\text{LOCp}(\text{M}_\text{r})$  and  $\text{LOCq}(\text{N}_\text{r})$  used in the argument descriptions above. For details, see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28 and “NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process” on page 868.
5. The global matrix  $A$  must be distributed using a square block-cyclic distribution; that is,  $\text{MB}_A = \text{NB}_A$ .
6. The global matrix  $A$  must be aligned on a block boundary; that is:
  - $ia-1$  must be a multiple of  $\text{MB}_A$
  - $ja-1$  must be a multiple of  $\text{NB}_A$
7. In the process grid, the process row containing the first row of the submatrix  $A$  must also contain the first row of the submatrix  $B$ ; that is:  $iarow = ibrow$  where:
  - $iarow = \text{mod}(\text{RSRC}_A + (ia-1)/\text{MB}_A, p)$
  - $ibrow = \text{mod}(\text{RSRC}_B + (ib-1)/\text{MB}_B, p)$
8. In the process grid, the process column containing the first column of the submatrix  $A$  must also contain the first column of the submatrix  $B$ ; that is:  $iacol = ibcol$  where:
  - $iacol = \text{mod}(\text{CSRC}_A + (ja-1)/\text{NB}_A, q)$
  - $ibcol = \text{mod}(\text{CSRC}_B + (jb-1)/\text{NB}_B, q)$
9. The block row offset of the global matrix  $A$  must be equal to the block row offset of the global matrix  $B$ ; that is:
  - $\text{mod}((ia-1, \text{MB}_A) = \text{mod}(ib-1, \text{MB}_B))$
10. The block column offset of the global matrix  $A$  must be equal to the block column offset of the global matrix  $B$ ; that is:
  - $\text{mod}((ja-1, \text{NB}_A) = \text{mod}(jb-1, \text{NB}_B))$
11. The following values must be equal:
  - $\text{MB}_A = \text{MB}_B$
  - $\text{NB}_A = \text{NB}_B$
  - $\text{CTXT}_A = \text{CTXT}_B$
12. For suggested block sizes, see “Coding Tips for Optimizing Parallel Performance” on page 77.

## Function

These subroutines reduce a real symmetric or complex Hermitian positive definite generalized Eigenproblem to standard form.

For PDSYGST:

- If  $ibtype = 1$ , the problem is  $Ax = \lambda Bx$ , and on output
  - $A = U^{-T}AU^{-1}$  if  $uplo = 'U'$ .
  - $A = L^{-1}AL^{-T}$  if  $uplo = 'L'$ .
- If  $ibtype = 2$ , the problem is  $ABx = \lambda x$ , and on output
  - $A = UAU^T$  if  $uplo = 'U'$ .
  - $A = L^TAL$  if  $uplo = 'L'$ .
- If  $ibtype = 3$ , the problem is  $BAx = \lambda x$ , and on output
  - $A = UAU^T$  if  $uplo = 'U'$ .
  - $A = L^TAL$  if  $uplo = 'L'$ .
- $B$  must have been previously factored by a call to PDPOTRF, as:

- $A = U^T U$  if  $uplo = 'U'$ .
- $A = LL^T$  if  $uplo = 'L'$ .

For PZHEGST:

- If  $ibtype = 1$ , the problem is  $Ax = \lambda Bx$ , and on output
  - $A = U^{-H} A U^{-1}$  if  $uplo = 'U'$ .
  - $A = L^{-1} A L^{-H}$  if  $uplo = 'L'$ .
- If  $ibtype = 2$ , the problem is  $ABx = \lambda x$ , and on output
  - $A = U A U^H$  if  $uplo = 'U'$ .
  - $A = L^H A L$  if  $uplo = 'L'$ .
- If  $ibtype = 3$ , the problem is  $B A x = \lambda x$ , and on output
  - $A = U A U^H$  if  $uplo = 'U'$ .
  - $A = L^H A L$  if  $uplo = 'L'$ .
- $B$  must have been previously factored by a call to PZPOTRF, as:
  - $A = U^H U$  if  $uplo = 'U'$ .
  - $A = LL^H$  if  $uplo = 'L'$ .

In the formulas above:

- $A$  represents the global submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$
- $B$  represents the global submatrix  $B_{ib:ib+n-1, jb:jb+n-1}$
- $L$  is a lower triangular matrix.
- $U$  is an upper triangular matrix.

## Error Conditions

### Computational Errors

None

### Resource Errors

None

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. DTYPE\_A is invalid.
2. DTYPE\_B is invalid.

#### Stage 2:

1. CTXT\_A is invalid.

#### Stage 3:

1. This subroutine has been called from outside the process grid.

#### Stage 4:

1.  $ibtype \neq 1, 2, \text{ or } 3$
2.  $uplo \neq 'U' \text{ or } 'L'$
3.  $n < 0$
4.  $M\_A < 0$  and  $n = 0$ ;  $M\_A < 1$  otherwise
5.  $N\_A < 0$  and  $n = 0$ ;  $N\_A < 1$  otherwise
6.  $MB\_A < 1$
7.  $NB\_A < 1$
8.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
9.  $CSRC\_A < 0$  or  $CSRC\_A \geq q$

10.  $ia < 1$
11.  $ja < 1$
12.  $M\_B < 0$  and  $n = 0$ ;  $M\_B < 1$  otherwise
13.  $N\_B < 0$  and  $n = 0$ ;  $N\_B < 1$  otherwise
14.  $MB\_B < 1$
15.  $NB\_B < 1$
16.  $RSRC\_B < 0$  or  $RSRC\_B \geq p$
17.  $CSRC\_B < 0$  or  $CSRC\_B \geq q$
18.  $ib < 1$
19.  $jb < 1$
20.  $CTXT\_A \neq CTXT\_B$

**Stage 5:** If  $n \neq 0$ :

1.  $ia > M\_A$
2.  $ja > N\_A$
3.  $ia+n-1 > M\_A$
4.  $ja+n-1 > N\_A$
5.  $ib > M\_B$
6.  $jb > N\_B$
7.  $ib+n-1 > M\_B$
8.  $jb+n-1 > N\_B$

In all cases:

1.  $MB\_A \neq NB\_A$
2.  $\text{mod}(ia-1, MB\_A) \neq 0$
3.  $\text{mod}(ja-1, NB\_A) \neq 0$
4.  $MB\_A \neq MB\_B$
5.  $NB\_A \neq NB\_B$
6.  $\text{mod}(ib-1, MB\_B) \neq \text{mod}(ia-1, MB\_A)$
7.  $\text{mod}(jb-1, NB\_B) \neq \text{mod}(ja-1, NB\_A)$
8. In the process grid, the process row containing the first row of the submatrix  $A$  does not contain the first row of the submatrix  $B$ ; that is,  $iarow \neq ibrow$ , where:
  - $iarow = \text{mod}(RSRC\_A + (ia-1)/MB\_A, p)$
  - $ibrow = \text{mod}(RSRC\_B + (ib-1)/MB\_B, p)$
9. In the process grid, the process column containing the first column of the submatrix  $A$  does not contain the first column of the submatrix  $B$ ; that is,  $iacol \neq ibcol$ , where:
  - $iacol = \text{mod}(CSRC\_A + (ja-1)/NB\_A, q)$
  - $ibcol = \text{mod}(CSRC\_B + (jb-1)/NB\_B, q)$

**Stage 6:**

1.  $LLD\_A < \max(1, \text{LOCp}(M\_A))$
2.  $LLD\_B < \max(1, \text{LOCp}(M\_B))$

**Stage 7:**

Each of the following global input arguments are checked to determine whether its value differs from the value specified on process  $P_{00}$ :

1.  $ibtype$  differs.
2.  $uplo$  differs.
3.  $n$  differs.
4.  $ia$  differs.
5.  $ja$  differs.
6.  $DTYPE\_A$  differs.
7.  $M\_A$  differs.
8.  $N\_A$  differs.

## PDSYGST and PZHEGST

9. MB\_A differs.
10. NB\_A differs.
11. RSRC\_A differs.
12. CSRC\_A differs.
13. *ib* differs.
14. *jb* differs.
15. DTYPE\_B differs.
16. M\_B differs.
17. N\_B differs.
18. MB\_B differs.
19. NB\_B differs.
20. RSRC\_B differs.
21. CSRC\_B differs.

## Examples

### Example 1

This example shows the reduction of a real symmetric positive definite generalized eigenproblem to standard form, using a  $2 \times 2$  process grid.

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      UPLO  N   B   IB   JB   DESCB  INFO
      |    |   |   |   |   |    |
CALL PDPOTRF( 'L', 4, B, 1, 1, DESCB, INFO )

      IBTYPE  UPLO  N   A   IA   JA   DESCA  B   IB   JB   DESCB  SCALE  INFO
      |       |    |   |   |   |   |    |   |   |   |   |    |
CALL PDSYGST( 1,  'L', 4, A, 1, 1, DESCA, B, 1, 1, DESCB, SCALE, INFO )
```

	DESC_A	DESC_B
DTYPE_	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	4	4
N_	4	4
MB_	1	1
NB_	1	1
RSRC_	0	0
CSRC_	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>

#### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:

```
LLD_A = MAX(1, NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_B = MAX(1, NUMROC(M_B, MB_B, MYROW, RSRC_B, NPROW))
```

In this example,  $LLD\_A$  and  $LLD\_B = 2$  on all processes.

Global real symmetric matrix  $A$  of order 4, stored in lower storage mode, with block sizes  $1 \times 1$ :

B,D	0	1	2	3
0	-1.0	.	.	.
1	1.0	1.0	.	.
2	-1.0	-1.0	1.0	.
3	1.0	1.0	-1.0	1.0

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$
3		

Local arrays for  $A$ :

p,q	0	1
0	-1.0 . -1.0 1.0	. . -1.0 .
1	1.0 . 1.0 -1.0	1.0 . 1.0 1.0

**Input to PDPOTRF:**

Global real symmetric positive definite matrix  $B$  of order 4, stored in lower storage mode, with block sizes  $1 \times 1$ :

B,D	0	1	2	3
0	2.0	.	.	.
1	1.0	2.0	.	.
2	0.0	1.0	2.0	.
3	0.0	0.0	1.0	2.0

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$
3		

Local arrays for  $B$ :

p,q	0	1
0	2.0 .	. .

## PDSYGST and PZHEGST

	0.0	2.0	1.0	.
1	1.0	.	2.0	.
	0.0	1.0	0.0	2.0

### Output from PDPOTRF and input to PDSYGST:

Global real symmetric positive definite matrix  $B$  of order 4, stored in lower storage mode, with block sizes  $1 \times 1$ :

B,D	0	1	2	3
0	1.4142	.	.	.
1	0.7071	1.2247	.	.
2	0.0000	0.8165	1.1547	.
3	0.0000	0.0000	0.8660	1.1180

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$
3		

Local arrays for  $B$ :

p,q	0	1
0	1.4142 0.0000	. 1.1547
1	0.7071 0.0000	. 0.8660

### Output from PDSYGST:

Global real symmetric matrix  $A$  of order 4, stored in lower storage mode, with block sizes  $1 \times 1$ :

B,D	0	1	2	3
0	-0.5000	.	.	.
1	0.8660	-0.1667	.	.
2	-1.2247	-0.2357	1.1667	.
3	1.5811	0.5477	-1.9365	3.1000

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$
3		

Local arrays for A:

p,q	0		1	
0	-0.5000	1.1667	-0.2357	1.1000
1	0.8660	-1.9365	-0.1667	3.1000

The value of *scale* is 1.0 on all processes.

The value of *info* is 0 on all processes.

## Example 2

This example shows the reduction of a complex Hermitian positive definite generalized eigenproblem to standard form, using a  $2 \times 2$  process grid.

### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      UPLO N  B  IB  JB  DESCB  INFO
CALL PZPOTRF( 'L', 4, B, 1, 1, DESCB, INFO )

      IBTYPE  UPLO  N  A  IA  JA  DESCA  B  IB  JB  DESCB  SCALE  INFO
CALL PZHEGST( 1, 'L', 4, A, 1, 1, DESCA, B, 1, 1, DESCB, SCALE, INFO )
```

	DESC_A	DESC_B
DTYPE_	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	4	4
N_	4	4
MB_	1	1
NB_	1	1
RSRC_	0	0
CSRC_	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>

### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.

2. Each process should set the LLD\_ as follows:

```
LLD_A = MAX(1, NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_B = MAX(1, NUMROC(M_B, MB_B, MYROW, RSRC_B, NPROW))
```

In this example, LLD\_A and LLD\_B = 2 on all processes.

Global complex Hermitian matrix *A* of order 4, stored in lower storage mode, with block sizes  $1 \times 1$ :

## PDSYGST and PZHEGST

B,D	0	1	2	3
0	(1.0, 0.0)	.	.	.
1	(5.0, -2.0)	(10.0, 0.0)	.	.
2	(7.0, 4.0)	(15.0, 6.0)	(20.0, 0.0)	.
3	(9.0, -6.0)	(20.0, -4.0)	(25.0, -9.0)	(30.0, 0.0)

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$
3		

Local arrays for *A*:

p,q	0	1
0	( 1.0, . ) ( 7.0, 4.0) (20.0, . )	(15.0, 6.0) .
1	( 5.0, -2.0) . ( 9.0, -6.0) (25.0, -9.0)	(10.0, . ) . (20.0, -4.0) (30.0, . )

**Input to PZPOTRF:**

Global complex Hermitian positive definite matrix *B* of order 4, stored in lower storage mode, with block sizes  $1 \times 1$ :

B,D	0	1	2	3
0	(9.0, 0.0)	.	.	.
1	(3.0, -3.0)	(18.0, 0.0)	.	.
2	(3.0, 3.0)	( 8.0, 6.0)	(27.0, 0.0)	.
3	(3.0, -3.0)	( 8.0, -6.0)	(12.0, -9.0)	(61.0, 0.0)

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$
3		

Local arrays for *B*:

p,q	0	1
0	(9.0, . ) (3.0, 3.0) (27.0, . )	( 8.0, 6.0) .
1	(3.0, -3.0) . (3.0, -3.0) (12.0, -9.0)	(18.0, . ) . ( 8.0, -6.0) (61.0, . )



**Output from PZPOTRF and input to PZHEGST:**

Global complex Hermitian positive definite matrix  $B$  of order 4, stored in lower storage mode, with block sizes  $1 \times 1$ :

B,D	0	1	2	3
0	(3.0, 0.0)	.	.	.
1	(1.0, -1.0)	(4.0, 0.0)	.	.
2	(1.0, 1.0)	(2.0, 0.0)	(4.4721, 0.0)	.
3	(1.0, -1.0)	(1.5, -1.5)	(2.3479, -.5590)	(6.9767, 0.0)

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$
3		

Local arrays for  $B$ :

p,q	0	1
0	(3.0, 0.0) (1.0, 1.0) (4.4721, 0.0)	(2.0, 1.0) .
1	(1.0, -1.0) . (1.0, -1.0) (2.3479, -.5590)	(4.0, 0.0) . (1.5, -1.5) (6.9767, 0.0)

**Output from PZHEGST:**

Global complex Hermitian matrix  $A$  of order 4, stored in lower storage mode, with block sizes  $1 \times 1$ :

B,D	0	1	2	3
0	(.1111, 0.0 )	.	.	.
1	(.3889, -.1389)	(.3472, 0.0 )	.	.
2	(.2919, .2485)	(.5714, -.0652)	(.0757, 0.0)	.
3	(.2422, -.2175)	(.2001, -.0009)	(.4003, .2645)	(-.0488, 0.0)

The following is the  $2 \times 2$  process grid:

B,D	0 2	1 3
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$
3		

Local arrays for  $A$ :

p,q	0	1
0	(.1111, 0.0 ) (.2919, .2485) (.0757, 0.0)	(.5714, -.0652) .

PDSYGST and PZHEGST

-----	----- -----
1	(.3889, -.1389) .   (.3472, 0.0 ) .
	(.2442, -.2175) (.4003, .2645)   (.2001, -.0009) (-.0488, 0.0 )

The value of *scale* is 1.0 on all processes.

The value of *info* is 0 on all processes.

## PDGEHRD — Reduce a General Matrix to Upper Hessenberg Form

### Purpose

This subroutine reduces a real general matrix  $A$  to upper Hessenberg form  $H$  by an orthogonal similarity transformation:

$$\bullet \quad H = Q^T A Q$$

where  $A$  represents the global general submatrix  $A_{ia+ilo-1: ia+ihi-1, ja+ilo-1: ja+ihi-1}$ .

If  $n = 0$ , no computation is performed, and the subroutine returns after doing some parameter checking. Then, if  $ihi = ilo$ , the subroutine returns after doing some parameter checking and setting  $\tau_{ja:ja+ilo-2}$  and  $\tau_{ja+ihi-1:ja+n-2}$  to zero.

See references [13] and [22].

Table 120. Data Types

$A, \tau, work$	Subroutine
Long-precision real	PDGEHRD

### Syntax

<b>Fortran</b>	CALL PDGEHRD ( $n, ilo, ihi, a, ia, ja, desc\_a, tau, work, lwork, info$ )
<b>C and C++</b>	pdgehrd ( $n, ilo, ihi, a, ia, ja, desc\_a, tau, work, lwork, info$ );

### On Entry

$n$  is the order of submatrix  $A$  used in the computation.

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

$ilo$  lower range of the rows or columns in the global general submatrix  $A$  used in the computation.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ilo \leq \max(1, n)$ .

$ihi$  upper range of the rows or columns in the global general submatrix  $A$  used in the computation.

Scope: **global**

Specified as: a fullword integer;  $\min(ilo, n) \leq ihi \leq n$ .

$a$  is the local part of the global general matrix  $A$ . This identifies the **first element** of the local array  $A$ . This subroutine computes the location of the first element of the local subarray used, based on  $ia, ja, desc\_a, p, q, myrow$ , and  $mycol$ ; therefore, the leading  $LOCp(ia+n-1)$  by  $LOCq(ja+n-1)$  part of the local array  $A$  must contain the local pieces of the leading  $ia+n-1$  by  $ja+n-1$  part of the global matrix.

Scope: **local**

Specified as: an  $LLD\_A$  by (at least)  $LOCq(N\_A)$  array, containing numbers of the data type indicated in Table 120. Details about the square block-cyclic data distribution of global matrix  $A$  are stored in  $desc\_a$ .

*ia* is the row index of the global matrix *A*, identifying the first row of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq M\_A$  and  $ia+n-1 \leq M\_A$ .

*ja* is the column index of the global matrix *A*, identifying the first column of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$  and  $ja+n-1 \leq N\_A$ .

*desc\_a* is the array descriptor for global matrix *A*, described in the following table:

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	If $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_A < p$	Global
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < q$	Global
9	LLD_A	The leading dimension of the local array	$LLD\_A \geq \max(1, LOCp(M\_A))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

*tau* See On Return.

*work* has the following meaning:

If  $lwork = 0$ , *work* is ignored.

If  $lwork \neq 0$ , *work* is the work area used by this subroutine, where:

- If  $lwork \neq -1$ , its size is (at least) of length *lwork*.
- If  $lwork = -1$ , its size is (at least) of length 1.

Scope: **local**

Specified as: an area of storage containing numbers of data type indicated in Table 120 on page 753.

*lwork* is the number of elements in array WORK.

Scope:

- If  $lwork \geq 0$ , *lwork* is **local**

- If  $lwork = -1$ ,  $lwork$  is **global**

Specified as: a fullword integer; where:

- If  $lwork = 0$ , PDGEHRD dynamically allocates the work area used by this subroutine. The work area is deallocated before control is returned to the calling program. This option is an extension to the ScaLAPACK standard.
- If  $lwork = -1$ , PDGEHRD performs a work area query and returns the minimum size of  $work$  in  $work_1$ . No computation is performed and the subroutine returns after error checking is complete.
- Otherwise, it must have the following value:

$$lwork \geq (nb \times nb) + nb \times \max(ihip+1, ihlp+inlq)$$

where:

- $nb = MB\_A = NB\_A$
- $ioff = \text{mod}(ia+ilo-2, nb)$
- $iroffa = \text{mod}(ia-1, nb)$
- $iarow = \text{mod}(RSRC\_A+(ia-1)/nb, nprow)$
- $ilrow = \text{mod}(RSRC\_A+(ia+ilo-2)/nb, nprow)$
- $ilcol = \text{mod}(CSRC\_A+(ja+ilo-2)/nb, npcol)$
- $ihip = \text{NUMROC}(ihi+iroffa, nb, myrow, iarow, nprow)$
- $ihlp = \text{NUMROC}(ihi-ilo+ioff+1, nb, myrow, ilrow, nprow)$
- $inlq = \text{NUMROC}(n-ilo+ioff+1, nb, mycol, ilcol, npcol)$

*info* See On Return.

### On Return

*a* is the updated local part of the global general matrix  $A$ , containing the results of the computation.

The upper triangle and the first subdiagonal of  $A_{ia:ia+n-1, ja:ja+n-1}$  are overwritten by the corresponding elements of the upper Hessenberg matrix  $H$ . The elements below the first subdiagonal are overwritten with  $v_{i+2:ihi}$ . These elements with  $\tau$  represent the orthogonal matrix  $Q$  as a product of elementary reflectors.

See “Function” on page 756, for more information.

Scope: **local**

Returned as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 120 on page 753. Details about the square block-cyclic data distribution of global matrix  $A$  are stored in *desc\_a*.

*tau* is the updated local part of the global matrix  $\tau$ , where:

- $\tau_{ja+ilo-1:ja+ihi-2}$  contains the scalar factors of the elementary reflectors.
- $\tau_{ja:ja+ilo-2}$  are set to zero.
- $\tau_{ja+ihi-1:ja+n-2}$  are set to zero.

This identifies the **first element** of the local array  $\tau$ . This subroutine computes the location of the first element of the local subarray used, based on  $ja$ , *desc\_a*,  $p$ ,  $q$ ,  $myrow$ , and  $mycol$ ; therefore, the leading 1 by LOCq( $ja+n-2$ ) part of the local array  $\tau$  must contain the local pieces of the leading 1 by  $ja+n-2$  part of the global matrix  $\tau$ .

A copy of the vector  $\tau$ , with a block size of NB\_A and global index  $ja$ , is returned to each row of the process grid. The process column over which the first column of  $\tau$  is distributed is CSRC\_A.

Scope: **local**

Returned as: a 1 by (at least)  $\text{LOCq}(\text{N\_A}-1)$  array, containing numbers of the data type indicated in Table 120 on page 753.

*work* is the work area used by this subroutine if  $lwork \neq 0$ , where:

If  $lwork \neq 0$  and  $lwork \neq -1$ , its size is (at least) of length  $lwork$ .

If  $lwork = -1$ , its size is (at least) of length 1.

Scope: **local**

Returned as: an area of storage, where:

If  $lwork \geq 1$  or  $lwork = -1$ , then  $work_1$  is set to the minimum  $lwork$  value and contains numbers of the data type indicated in Table 120 on page 753. Except for  $work_1$ , the contents of  $work$  are overwritten on return.

*info* indicates that a successful computation occurred.

Scope: **global**

Returned as: a fullword integer;  $info = 0$ .

## Notes and Coding Rules

1. In your C program, argument *info* must be passed by reference.
2. Matrix *A*,  $\tau$ , and *work* must have no common elements; otherwise, results are unpredictable.
3. On entry, the general submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  must already be upper triangular in rows ( $ia:ia+ilo-2$ ) and ( $ia+ihi:ia+n-1$ ), and upper triangular in columns ( $ja:ja+ilo-2$ ) and ( $ja+ihi:ja+n-1$ ). If this is not the case, you should set  $ilo = 1$  and  $ihi = n$ .  
If  $n = 0$ , you should set  $ilo = 1$  and  $ihi = 0$ . If  $n > 0$ , you should set  $1 \leq ilo \leq ihi \leq n$ .
4. The NUMROC utility subroutine can be used to determine the values of  $\text{LOCp}(\text{M\_})$  and  $\text{LOCq}(\text{N\_})$  used in the argument descriptions above. For details, see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28 and “NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process” on page 868.
5. The global general matrix *A* must be distributed using a square block-cyclic distribution; that is,  $\text{MB\_A} = \text{NB\_A}$ .
6. The global general matrix *A* must be aligned on a block boundary; that is:
  - $ia-1$  must be a multiple of  $\text{MB\_A}$
  - $ja-1$  must be a multiple of  $\text{NB\_A}$
7. There is no array descriptor for  $\tau$ .  $\tau$  is a row-distributed vector with block size  $\text{NB\_A}$ , local arrays of dimension 1 by  $\text{LOCq}(\text{N\_A}-1)$ , and global index *ja*. A copy of  $\tau$  exists on each row of the process grid, and the process column over which the first column of  $\tau$  is distributed is  $\text{CSRC\_A}$ .
8. For suggested block sizes, see “Coding Tips for Optimizing Parallel Performance” on page 77.
9. If  $lwork = -1$  on any process, it must equal  $-1$  on all processes. That is, if a subset of the processes specifies  $-1$  for the work area size, they must all specify  $-1$ .

## Function

This subroutine reduces a real general matrix *A* to upper Hessenberg form *H* by an orthogonal similarity transformation:

- $H = Q^T A Q$

where:

- $A$  represents the global general submatrix  $A_{ia+ilo-1:ia+ihi-1, ja+ilo-1:ja+ihi-1}$
- Matrix  $Q$  is represented as a product of  $(ihi-ilo)$  elementary reflectors:
  - $Q = H_{ilo} H_{ilo+1} \dots H_{ihi-1}$

where:

- For each  $i$ :  $H_i = I - \tau v v^T$
- $\tau$  is a real scalar
- $v$  is a real vector with  $v_{1:i} = 0$ ,  $v_{i+1} = 1$ , and  $v_{ihi+1:n} = 0$
- $v_{i+2:ihi}$  is stored on return in in submatrix  $A_{i+ilo+1+(ia-1):ihi+(ia-1), ilo+i-1+(ja-1)}$
- $\tau$  is stored on return in  $\tau_{i+ilo-1+(ja-1)}$
- $I$  is the identity matrix

The following example shows the contents of the general submatrix  $A$  on entry with  $n = 7$ ,  $ia = ja = 1$ ,  $ilo = 2$ , and  $ihi = 6$ :

$$\begin{bmatrix} a & a & a & a & a & a & a \\ . & a & a & a & a & a & a \\ . & a & a & a & a & a & a \\ . & a & a & a & a & a & a \\ . & a & a & a & a & a & a \\ . & a & a & a & a & a & a \\ . & . & . & . & . & . & a \end{bmatrix}$$

Following is the general submatrix  $A$  on return:

$$\begin{bmatrix} a & a & h & h & h & h & a \\ . & a & h & h & h & h & a \\ . & h & h & h & h & h & h \\ . & v_2 & h & h & h & h & h \\ . & v_2 & v_3 & h & h & h & h \\ . & v_2 & v_3 & v_4 & h & h & h \\ . & . & . & . & . & . & a \end{bmatrix}$$

where:

- $a$  represents an element of the original submatrix  $A$ .
- $h$  represents a updated element of the upper Hessenberg matrix  $H$ .
- $v_i$  represents the corresponding elements of the vector defining  $H_{ilo+i-1+(ja-1)}$ .

## Error Conditions

### Computational Errors

None

### Resource Errors

1.  $lwork = 0$  and unable to allocate work space

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1.  $DTYPE\_A$  is invalid.

#### Stage 2:

1.  $CTXT\_A$  is invalid.

**Stage 3:**

1. PDGEHRD has been called from outside the process grid.

**Stage 4:**

1.  $n < 0$
2.  $M\_A < 0$  and  $n = 0$ ;  $M\_A < 1$  otherwise
3.  $N\_A < 0$  and  $n = 0$ ;  $N\_A < 1$  otherwise
4.  $MB\_A < 1$
5.  $NB\_A < 1$
6.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
7.  $CSRC\_A < 0$  or  $CSRC\_A \geq q$
8.  $ia < 1$
9.  $ja < 1$

**Stage 5:**

1.  $ilo < 1$  or  $ilo > \max(1, n)$
2.  $ihi < \min(ilo, n)$  or  $ihi > n$

If  $n \neq 0$ :

1.  $ia > M\_A$
2.  $ja > N\_A$
3.  $ia+n-1 > M\_A$
4.  $ja+n-1 > N\_A$

In all cases:

1.  $MB\_A \neq NB\_A$
2.  $\text{mod}(ia-1, MB\_A) \neq 0$
3.  $\text{mod}(ja-1, NB\_A) \neq 0$

**Stage 6:**

1.  $LLD\_A < \max(1, \text{LOCp}(M\_A))$
2.  $lwork \neq 0$ ,  $lwork \neq -1$ , and  $lwork < (nb \times nb) + nb \times \max(ihip+1, ihlp+inlq)$

where:

- $nb = MB\_A = NB\_A$
- $ioff = \text{mod}(ia+ilo-2, nb)$
- $iroffa = \text{mod}(ia-1, nb)$
- $iarow = \text{mod}(RSRC\_A+(ia-1)/nb, nprow)$
- $ilrow = \text{mod}(RSRC\_A+(ia+ilo-2)/nb, nprow)$
- $icol = \text{mod}(CSRC\_A+(ja+ilo-2)/nb, npcot)$
- $ihip = \text{NUMROC}(ihi+iroffa, nb, myrow, iarow, nprow)$
- $ihlp = \text{NUMROC}(ihi-ilo+ioff+1, nb, myrow, ilrow, nprow)$
- $inlq = \text{NUMROC}(n-ilo+ioff+1, nb, mycol, icol, npcot)$

**Stage 7:**

Each of the following global input arguments are checked to determine whether its value differs from the value specified on process  $P_{00}$ :

1.  $n$  differs.
2.  $ilo$  differs.
3.  $ihi$  differs.
4.  $ia$  differs.
5.  $ja$  differs.
6.  $DTYPE\_A$  differs.
7.  $M\_A$  differs.
8.  $N\_A$  differs.
9.  $MB\_A$  differs.



10. NB\_A differs.
11. RSRC\_A differs.
12. CSRC\_A differs.

Also:

13.  $lwork = -1$  on a subset of processes.

## Examples

### Example

This example shows the reduction of a general matrix of order 3 to upper Hessenberg form using a  $2 \times 2$  process grid.

**Note:** Because  $lwork = 0$ , PDGEHRD dynamically allocates the work area used by this subroutine.

### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      N   ILO   IHI   A   IA   JA   DESC_A   TAU   WORK   LWORK   INFO
      |   |   |   |   |   |   |   |   |   |   |
CALL PDGEHRD( 3 , 1 , 3 , A , 1 , 1 , DESC_A , TAU , WORK , 0 , INFO)
```

	DESC_A
DTYPE_	1
CTXT_	<i>icontxt</i> <sup>1</sup>
M_	3
N_	3
MB_	1
NB_	1
RSRC_	0
CSRC_	0
LLD_	See below <sup>2</sup>

### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:  
 $LLD\_A = \text{MAX}(1, \text{NUMROC}(M\_A, MB\_A, MYROW, RSRC\_A, NPROW))$   
 In this example,  $LLD\_A = 2$  on  $P_{00}$  and  $P_{01}$ , and  $LLD\_A = 1$  on  $P_{10}$  and  $P_{11}$ .

Global general matrix  $A$  of order 3 with block sizes  $1 \times 1$ :

$$B, D \quad \begin{array}{ccc} & 0 & 1 & 2 \\ \begin{array}{c} 0 \\ 1 \\ 2 \end{array} & \left[ \begin{array}{c|c|c} 33.0 & 16.0 & 72.0 \\ \hline -24.0 & -10.0 & -57.0 \\ \hline -8.0 & -4.0 & -17.0 \end{array} \right] \end{array}$$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

Local arrays for  $A$ :

p,q	0	1
0	33.0 72.0 -8.0 -17.0	16.0 -4.0
1	-24.0 -57.0	-10.0

### Output:

Global general matrix  $A$  of order 3 with block sizes  $1 \times 1$ :

B,D	0	1	2
0	33.00	-37.95	63.25
1	25.30	-29.00	53.00
2	0.16	0.00	2.00

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

Local arrays for  $A$ :

p,q	0	1
0	33.0 63.25 0.16 2.00	-37.95 0.00
1	25.30 53.00	-29.00

Global row vector  $\tau$  of length 2 with block sizes of 1:

B,D	0	1
0	1.95	0.00

**Note:** A copy of  $\tau$  is distributed across each row of the process grid.

The following is the  $2 \times 2$  process grid:

B,D	0	1
	$P_{00}$	$P_{01}$
	$P_{10}$	$P_{11}$

Local arrays for  $\tau$ :

p,q	0	1
0	1.95	0.00
1	1.95	0.00

The value of *info* is 0 on all processes.

## PDGEBRD and PZGEBRD — Reduce a General Matrix to Bidiagonal Form

### Purpose

PDGEBRD reduces a real general matrix  $A$  of order  $m$  by  $n$  to upper or lower bidiagonal form  $B$  by an orthogonal transformation:

$$\bullet B = Q^T A P$$

where:

- $A$  represents the global real general submatrix  $A_{ia:ia+m-1, ja:ja+n-1}$ .
- If  $m \geq n$ , then  $B$  is upper bidiagonal.
- If  $m < n$ , then  $B$  is lower bidiagonal.

PZGEBRD reduces a complex general matrix  $A$  of order  $m$  by  $n$  to upper or lower bidiagonal form  $B$  by a unitary transformation:

$$\bullet B = Q^H A P$$

where:

- $A$  represents the global complex general submatrix  $A_{ia:ia+m-1, ja:ja+n-1}$ .
- If  $m \geq n$ , then  $B$  is upper bidiagonal.
- If  $m < n$ , then  $B$  is lower bidiagonal.

If  $\min(m, n) = 0$ , no computation is performed and the subroutine returns after doing some parameter checking.

See references [13] and [22].

Table 121. Data Types

$A, \tau_q, \tau_p, work$	$d, e$	Subroutine
Long-precision real	Long-precision real	PDGEBRD
Long-precision complex	Long-precision real	PZGEBRD

### Syntax

<b>Fortran</b>	CALL PDGEBRD   PZGEBRD ( $m, n, a, ia, ja, desc\_a, d, e, tauq, taup, work, lwork, info$ )
<b>C and C++</b>	pdgebrd   pzgebrd ( $m, n, a, ia, ja, desc\_a, d, e, tauq, taup, work, lwork, info$ );

### On Entry

$m$  is the number of rows of submatrix  $A$  used in the computation.

Scope: **global**

Specified as: a fullword integer;  $m \geq 0$

$n$  is the number of columns of submatrix  $A$  used in the computation.

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

$a$  is the local part of the global real or complex general matrix  $A$ . This identifies the **first element** of the local array  $A$ . This subroutine computes the location of the first element of the local subarray used, based on  $ia, ja, desc\_a, p, q, myrow$ , and  $mycol$ ; therefore, the leading  $LOCp(ia+m-1)$  by

$\text{LOCq}(ja+n-1)$  part of the local array  $A$  must contain the local pieces of the leading  $ia+m-1$  by  $ja+n-1$  part of the global matrix.

Scope: **local**

Specified as: an  $\text{LLD\_A}$  by (at least)  $\text{LOCq}(\text{N\_A})$  array, containing numbers of the data type indicated in Table 121 on page 762. Details about the square block-cyclic data distribution of global matrix  $A$  are stored in  $\text{desc\_a}$ .

$ia$  is the row index of the global matrix  $A$ , identifying the first row of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq \text{M\_A}$  and  $ia+m-1 \leq \text{M\_A}$ .

$ja$  is the column index of the global matrix  $A$ , identifying the first column of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq \text{N\_A}$  and  $ja+n-1 \leq \text{N\_A}$ .

$\text{desc\_a}$  is the array descriptor for global matrix  $A$ , described in the following table:

$\text{desc\_a}$	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	$\text{DTYPE\_A}=1$	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $m = 0$ or $n = 0$ : $\text{M\_A} \geq 0$ Otherwise: $\text{M\_A} \geq 1$	Global
4	N_A	Number of columns in the global matrix	If $m = 0$ or $n = 0$ : $\text{N\_A} \geq 0$ Otherwise: $\text{N\_A} \geq 1$	Global
5	MB_A	Row block size	$\text{MB\_A} \geq 1$	Global
6	NB_A	Column block size	$\text{NB\_A} \geq 1$	Global
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq \text{RSRC\_A} < p$	Global
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq \text{CSRC\_A} < q$	Global
9	LLD_A	The leading dimension of the local array	$\text{LLD\_A} \geq \max(1, \text{LOCp}(\text{M\_A}))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

$d$  See On Return.

$e$  See On Return.

$\text{tauq}$  See On Return.

$\text{taup}$  See On Return.

$\text{work}$  has the following meaning:

If  $\text{lwork} = 0$ ,  $\text{work}$  is ignored.

If  $lwork \neq 0$ ,  $work$  is the work area used by this subroutine, where:

- If  $lwork \neq -1$ , its size is (at least) of length  $lwork$ .
- If  $lwork = -1$ , its size is (at least) of length 1.

Scope: **local**

Specified as: an area of storage containing numbers of data type indicated in Table 121 on page 762.

$lwork$  is the number of elements in array  $WORK$ .

Scope:

- If  $lwork \geq 0$ ,  $lwork$  is **local**
- If  $lwork = -1$ ,  $lwork$  is **global**

Specified as: a fullword integer; where:

- If  $lwork = 0$ , PDGEBRD and PZGEBRD dynamically allocate the work area used by these subroutines. The work area is deallocated before control is returned to the calling program. This option is an extension to the ScaLAPACK standard.
- If  $lwork = -1$ , PDGEBRD and PZGEBRD perform a work area query and return the minimum size of  $work$  in  $work_1$ . No computation is performed and the subroutines return after error checking is complete.
- Otherwise,  $lwork$  must have the following value:

$$lwork \geq nb(mp0+nq0+1)+nq0$$

where:

- $nb = MB\_A = NB\_A$
- $iroffa = \text{mod}(ia-1, nb)$
- $icoffa = \text{mod}(ja-1, nb)$
- $iarow = \text{mod}(RSRC\_A+(ia-1)/nb, nprow)$
- $iacol = \text{mod}(CSRC\_A+(ja-1)/nb, npcol)$
- $mp0 = \text{NUMROC}(m+iroffa, nb, myrow, iarow, nprow)$
- $nq0 = \text{NUMROC}(n+icoffa, nb, mycol, iacol, npcol)$

*info* See On Return.

## On Return

$a$  is the updated local part of the global general matrix  $A$ , containing the results of the computation, where:

- If  $m \geq n$ , the diagonal and first superdiagonal of  $A_{ia:ia+m-1, ja:ja+n-1}$  are overwritten by the corresponding elements of the upper bidiagonal matrix  $B$ . The elements below the diagonal are overwritten with  $v_{i+1:m}$ . These elements with  $\tau_q$  represent the matrix  $Q$  as a product of elementary reflectors. The elements above the first superdiagonal are overwritten with  $u_{i+2:n}$ . These elements with  $\tau_p$  represent the matrix  $P$  as a product of elementary reflectors.
- If  $m < n$ , the diagonal and first subdiagonal of  $A_{ia:ia+m-1, ja:ja+n-1}$  are overwritten by the corresponding elements of the lower bidiagonal matrix  $B$ . The elements below the first subdiagonal are overwritten with  $v_{i+2:m}$ . These elements with  $\tau_q$  represent the matrix  $Q$  as a product of elementary reflectors. The elements above the diagonal are overwritten with  $u_{i+1:n}$ . These elements with  $\tau_p$  represent the matrix  $P$  as a product of elementary reflectors.

See "Function" on page 768, for more information.

Scope: **local**

Returned as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 121 on page 762. Details about the square block-cyclic data distribution of global matrix *A* are stored in *desc\_a*.

*d* is the updated local part of the global matrix *D*, where:

- If  $m \geq n$ , then  $d_{ja:ja+n-1}$  contains the diagonal elements of the bidiagonal matrix *B*.

This identifies the **first element** of the local array *D*. This subroutine computes the location of the first element of the local subarray used, based on *ja*, *desc\_a*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading 1 by LOCq(*ja+n-1*) part of the local array *D* must contain the local pieces of the leading 1 by *ja+n-1* part of the global matrix *D*.

A copy of the vector *d*, with a block size of NB\_A and global index *ja*, is returned to each row of the process grid. The process column over which the first column of *d* is distributed is CSRC\_A.

- If  $m < n$ , then  $d_{ia:ia+m-1}$  contains the diagonal elements of the bidiagonal matrix *B*.

This identifies the **first element** of the local array *D*. This subroutine computes the location of the first element of the local subarray used, based on *ia*, *desc\_a*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading LOCp(*ia+m-1*) by 1 part of the local array *D* must contain the local pieces of the leading *ia+m-1* by 1 part of the global matrix *D*.

A copy of the vector *d*, with a block size of MB\_A and global index *ia*, is returned to each column of the process grid. The process row over which the first row of *d* is distributed is RSRC\_A.

Scope: **local**

Returned as: a 1 by (at least) LOCq(N\_A) array if  $m \geq n$ , and a LOCp(M\_A) by 1 array if  $m < n$ , containing numbers of the data type indicated in Table 121 on page 762.

*e* is the updated local part of the global matrix *E*, where:

- If  $m \geq n$ , then  $e_{ia:ia+n-2}$  contains the superdiagonal elements of the bidiagonal matrix *B*.

This identifies the **first element** of the local array *E*. This subroutine computes the location of the first element of the local subarray used, based on *ia*, *desc\_a*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading LOCp(*ia+n-2*) by 1 part of the local array *E* must contain the local pieces of the leading *ia+n-2* by 1 part of the global matrix *E*.

A copy of the vector *e*, with a block size of MB\_A and global index *ia*, is returned to each column of the process grid. The process row over which the first row of *e* is distributed is RSRC\_A.

- If  $m < n$ , then  $e_{ja:ja+m-2}$  contains the subdiagonal elements of the bidiagonal matrix *B*.

This identifies the **first element** of the local array *D*. This subroutine computes the location of the first element of the local subarray used, based on *ja*, *desc\_a*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading 1 by LOCq(*ja+m-2*) part of the local array *E* must contain the local pieces of the leading 1 by *ja+m-2* part of the global matrix *E*.

A copy of the vector *e*, with a block size of NB\_A and global index *ja*, is returned to each row of the process grid. The process column over which the first column of *e* is distributed is CSRC\_A.

Scope: **local**

Returned as: an (at least) LOCp(N\_A-1) by 1 array if  $m \geq n$  and a 1 by (at least) LOCq(M\_A-1) array if  $m < n$ , containing numbers of the data type indicated in Table 121 on page 762.

*taug* is the updated local part of the global matrix  $\tau_q$ , where:

$$\tau_{q_{ja:ja+\min(m,n)-1}}$$

contains the scalar factors of the elementary reflectors which represent the matrix  $Q$ . See “Function” on page 768 for more details.

This identifies the **first element** of the local array  $\tau_q$ . This subroutine computes the location of the first element of the local subarray used, based on *ja*, *desc\_a*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading 1 by LOCq(*ja*+min(*m*, *n*)-1) part of the local array  $\tau_q$  must contain the local pieces of the leading 1 by *ja*+min(*m*, *n*)-1 part of the global matrix  $\tau_q$ .

A copy of the vector  $\tau_q$ , with a block size of NB\_A and global index *ja*, is returned to each row of the process grid. The process column over which the first column of  $\tau_q$  is distributed is CSRC\_A.

Scope: **local**

Returned as: a 1 by (at least) LOCq(min(M\_A, N\_A)) array, containing numbers of the data type indicated in Table 121 on page 762.

*taup* is the updated local part of the global matrix  $\tau_p$ , where:

$$\tau_{p_{ia:ia+\min(m,n)-1}}$$

contains the scalar factors of the elementary reflectors which represent the matrix  $P$ . See “Function” on page 768 for more details.

This identifies the **first element** of the local array  $\tau_p$ . This subroutine computes the location of the first element of the local subarray used, based on *ia*, *desc\_a*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading LOCp(*ia*+min(*m*, *n*)-1) by 1 part of the local array  $\tau_p$  must contain the local pieces of the leading *ia*+min(*m*, *n*)-1 by 1 part of the global matrix  $\tau_p$ .

A copy of the vector  $\tau_p$ , with a block size of MB\_A and global index *ia*, is returned to each column of the process grid. The process row over which the first row of  $\tau_p$  is distributed is RSRC\_A.

Scope: **local**

Returned as: an (at least) LOCp(min(M\_A, N\_A)) by 1 array, containing numbers of the data type indicated in Table 121 on page 762.

*work* is the work area used by this subroutine if *lwork*  $\neq$  0, where:

If *lwork*  $\neq$  0 and *lwork*  $\neq$  -1, its size is (at least) of length *lwork*.

If *lwork* = -1, its size is (at least) of length 1.

Scope: **local**



Returned as: an area of storage, where:

If  $lwork \geq 1$  or  $lwork = -1$ , then  $work_1$  is set to the minimum  $lwork$  value and contains numbers of the data type indicated in Table 121 on page 762. Except for  $work_1$ , the contents of  $work$  are overwritten on return.

*info* indicates that a successful computation occurred.

Scope: **global**

Returned as: a fullword integer; *info* = 0.

## Notes and Coding Rules

1. In your C program, argument *info* must be passed by reference.
2.  $A$ ,  $d$ ,  $e$ ,  $\tau_q$ ,  $\tau_p$ , and  $work$  must have no common elements; otherwise, results are unpredictable.
3. The NUMROC utility subroutine can be used to determine the values of  $LOCp(M\_)$  and  $LOCq(N\_)$  used in the argument descriptions above. For details, see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28 and “NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process” on page 868.
4. The global general matrix  $A$  must be distributed using a square block-cyclic distribution; that is,  $MB\_A = NB\_A$ .
5. For the global general matrix  $A$ , the block row offset must be equal to the block column offset; that is,  $\text{mod}(ia-1, MB\_A) = \text{mod}(ja-1, NB\_A)$ .
6. For suggested block sizes, see “Coding Tips for Optimizing Parallel Performance” on page 77.
7. There is no array descriptor for  $d$ , where:
  - If  $m \geq n$ , then  $d$  is a row-distributed vector with block size  $NB\_A$ , local array of dimension 1 by  $LOCq(N\_A)$ , and global index  $ja$ . A copy of  $d$  exists on each row of the process grid, and the process column over which the first column of  $d$  is distributed is  $CSRC\_A$ .
  - If  $m < n$ , then  $d$  is a column-distributed vector with block size  $MB\_A$ , local array of dimension  $LOCp(M\_A)$  by 1, and global index  $ia$ . A copy of  $d$  exists on each column of the process grid, and the process row over which the first row of  $d$  is distributed is  $RSRC\_A$ .
8. There is no array descriptor for  $e$ , where:
  - If  $m \geq n$ , then  $e$  is a column-distributed vector with block size  $MB\_A$ , local array of dimension  $LOCp(N\_A-1)$  by 1, and global index  $ia$ . A copy of  $e$  exists on each column of the process grid, and the process row over which the first row of  $e$  is distributed is  $RSRC\_A$ .
  - If  $m < n$ , then  $e$  is a row-distributed vector with block size  $NB\_A$ , local array of dimension 1 by  $LOCq(M\_A-1)$ , and global index  $ja$ . A copy of  $e$  exists on each row of the process grid, and the process column over which the first column of  $e$  is distributed is  $CSRC\_A$ .
9. There is no array descriptor for  $\tau_q$ .  $\tau_q$  is a row-distributed vector with block size  $NB\_A$ , local array of dimension 1 by  $LOCq(\min(M\_A, N\_A))$ , and global index  $ja$ . A copy of  $\tau_q$  exists on each row of the process grid, and the process column over which the first column of  $\tau_q$  is distributed is  $CSRC\_A$ .
10. There is no array descriptor for  $\tau_p$ .  $\tau_p$  is a column-distributed vector with block size  $MB\_A$ , local array of dimension  $LOCp(\min(M\_A, N\_A))$  by 1, and global index  $ia$ . A copy of  $\tau_p$  exists on each column of the process grid, and the process row over which the first row of  $\tau_p$  is distributed is  $RSRC\_A$ .

11. If  $lwork = -1$  on any process, it must equal  $-1$  on all processes. That is, if a subset of the processes specifies  $-1$  for the work area size, they must all specify  $-1$ .

## Function

### PDGEBRD

reduces a real general matrix  $A$  of order  $m$  by  $n$  to upper or lower bidiagonal form  $B$  by an orthogonal transformation:

$$\bullet B = Q^T A P$$

where:

- $A$  represents the global real general submatrix  $A_{ia:ia+m-1, ja:ja+n-1}$ .
- If  $m \geq n$ , then  $B$  is upper bidiagonal, and matrices  $Q$  and  $P$  are represented as the product of elementary reflectors:
  - $Q = H_1 H_2 \dots H_n$
  - $P = G_1 G_2 \dots G_{n-1}$

where:

- For each  $i$ :  $H_i = I - \tau_q v v^T$
- For each  $i$ :  $G_i = I - \tau_p u u^T$
- $\tau_q$  is a real scalar and is stored on return in:

$$\tau_{q_{i+(ja-1)}}$$

- $\tau_p$  is a real scalar and is stored on return in:

$$\tau_{p_{i+(ia-1)}}$$

- $v$  is a real vector with  $v_{1:i-1} = 0$  and  $v_i = 1$
- $v_{i+1:m}$  is stored on return in submatrix  $A_{i+1+(ia-1):m+(ia-1), i+(ja-1)}$
- $u$  is a real vector with  $u_{1:i} = 0$  and  $u_{i+1} = 1$
- $u_{i+2:n}$  is stored on return in submatrix  $A_{i+(ia-1), i+2+(ja-1):n+(ja-1)}$
- $I$  is the identity matrix

The following example shows the contents of  $A$  on return with  $ia = ja = 1$ ,  $m = 6$ , and  $n = 5$ :

$$\begin{bmatrix} d & e & u_1 & u_1 & u_1 \\ v_1 & d & e & u_2 & u_2 \\ v_1 & v_2 & d & e & u_3 \\ v_1 & v_2 & v_3 & d & e \\ v_1 & v_2 & v_3 & v_4 & d \\ v_1 & v_2 & v_3 & v_4 & v_5 \end{bmatrix}$$

where:

- $d$  represents the diagonal elements of  $B$
- $e$  represents the off-diagonal elements of  $B$
- $v_i$  represents the corresponding elements of the vector defining  $H_i$ .
- $u_i$  represents the corresponding elements of the vector defining  $G_i$ .
- If  $m < n$ , then  $B$  is lower bidiagonal, and matrices  $Q$  and  $P$  are represented as the product of elementary reflectors:
  - $Q = H_1 H_2 \dots H_{m-1}$

$$- P = G_1 G_2 \dots G_m$$

where:

- For each  $i$ :  $H_i = I - \tau_q v v^T$
- For each  $i$ :  $G_i = I - \tau_p u u^T$
- $\tau_q$  and  $\tau_p$  are real scalars
- $\tau_q$  is stored on return in:

$$\tau_{q_{i+(ja-1)}}$$

- $\tau_p$  is stored on return in:

$$\tau_{p_{i+(ia-1)}}$$

- $v$  is a real vector with  $v_{1:i} = 0$  and  $v_{i+1} = 1$
- $v_{i+2:m}$  is stored on return in submatrix  $A_{i+2+(ia-1):m+(ia-1), i+(ja-1)}$
- $u$  is a real vector with  $u_{1:i-1} = 0$  and  $u_i = 1$
- $u_{i+1:n}$  is stored on return in submatrix  $A_{i+(ia-1), i+1+(ja-1):n+(ja-1)}$
- $I$  is the identity matrix

The following example shows the contents of  $A$  on return with  $ia = ja = 1$ ,  $m = 5$ , and  $n = 6$ :

$$\begin{bmatrix} d & u_1 & u_1 & u_1 & u_1 & u_1 \\ e & d & u_2 & u_2 & u_2 & u_2 \\ v_1 & e & d & u_3 & u_3 & u_3 \\ v_1 & v_2 & e & d & u_4 & u_4 \\ v_1 & v_2 & v_3 & e & d & u_5 \end{bmatrix}$$

where:

- $d$  represents the diagonal elements of  $B$
- $e$  represents the off-diagonal elements of  $B$
- $v_i$  represents the corresponding elements of the vector defining  $H_i$ .
- $u_i$  represents the corresponding elements of the vector defining  $G_i$ .

### PZGEBRD

reduces a complex general matrix  $A$  of order  $m$  by  $n$  to upper or lower bidiagonal form  $B$  by a unitary transformation:

$$\bullet B = Q^H A P$$

where:

- $A$  represents the global complex general submatrix  $A_{ia:ia+m-1, ja:ja+n-1}$ .
- If  $m \geq n$ , then  $B$  is upper bidiagonal, and matrices  $Q$  and  $P$  are represented as the product of elementary reflectors:
  - $Q = H_1 H_2 \dots H_n$
  - $P = G_1 G_2 \dots G_{n-1}$

where:

- For each  $i$ :  $H_i = I - \tau_q v v^H$
- For each  $i$ :  $G_i = I - \tau_p u u^H$
- $\tau_q$  is a complex scalar and is stored on return in:

$$\tau_{q_{i+(ja-1)}}$$

- $\tau_p$  is a complex scalar and is stored on return in:

$$\tau_{p_{i+(ia-1)}}$$

- $v$  is a complex vector with  $v_{1:i-1} = (0, 0)$  and  $v_i = (1, 0)$
- $v_{i+1:m}$  is stored on return in submatrix  $A_{i+1+(ia-1):m+(ia-1), i+(ja-1)}$
- $u$  is a complex vector with  $u_{1:i} = (0, 0)$  and  $u_{i+1} = (1, 0)$
- $u_{i+2:n}$  is stored on return in submatrix  $A_{i+(ia-1), i+2+(ja-1):n+(ja-1)}$
- $I$  is the identity matrix

The following example shows the contents of  $A$  on return with  $ia = ja = 1$ ,  $m = 6$ , and  $n = 5$ :

$$\begin{bmatrix} d & e & u_1 & u_1 & u_1 \\ v_1 & d & e & u_2 & u_2 \\ v_1 & v_2 & d & e & u_3 \\ v_1 & v_2 & v_3 & d & e \\ v_1 & v_2 & v_3 & v_4 & d \\ v_1 & v_2 & v_3 & v_4 & v_5 \end{bmatrix}$$

where:

- $d$  represents the diagonal elements of  $B$
- $e$  represents the off-diagonal elements of  $B$
- $v_i$  represents the corresponding elements of the vector defining  $H_i$ .
- $u_i$  represents the corresponding elements of the vector defining  $G_i$ .
- If  $m < n$ , then  $B$  is lower bidiagonal, and matrices  $Q$  and  $P$  are represented as the product of elementary reflectors:
  - $Q = H_1 H_2 \dots H_{m-1}$
  - $P = G_1 G_2 \dots G_m$

where:

- For each  $i$ :  $H_i = I - \tau_q v v^H$
- For each  $i$ :  $G_i = I - \tau_p u u^H$
- $\tau_q$  and  $\tau_p$  are complex scalars
- $\tau_q$  is stored on return in:

$$\tau_{q_{i+(ja-1)}}$$

- $\tau_p$  is stored on return in:

$$\tau_{p_{i+(ia-1)}}$$

- $v$  is a complex vector with  $v_{1:i} = (0, 0)$  and  $v_{i+1} = (1, 0)$
- $v_{i+2:m}$  is stored on return in submatrix  $A_{i+2+(ia-1):m+(ia-1), i+(ja-1)}$
- $u$  is a complex vector with  $u_{1:i-1} = (0, 0)$  and  $u_i = (1, 0)$
- $u_{i+1:n}$  is stored on return in submatrix  $A_{i+(ia-1), i+1+(ja-1):n+(ja-1)}$
- $I$  is the identity matrix

The following example shows the contents of  $A$  on return with  $ia = ja = 1$ ,  $m = 5$ , and  $n = 6$ :

$$\begin{bmatrix} d & u_1 & u_1 & u_1 & u_1 & u_1 \\ e & d & u_2 & u_2 & u_2 & u_2 \\ v_1 & e & d & u_3 & u_3 & u_3 \\ v_1 & v_2 & e & d & u_4 & u_4 \\ v_1 & v_2 & v_3 & e & d & u_5 \end{bmatrix}$$

where:

- $d$  represents the diagonal elements of  $B$
- $e$  represents the off-diagonal elements of  $B$
- $v_i$  represents the corresponding elements of the vector defining  $H_i$ .
- $u_i$  represents the corresponding elements of the vector defining  $G_i$ .

## Error Conditions

### Computational Errors

None

### Resource Errors

1.  $lwork = 0$  and unable to allocate work space

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1.  $DTYPE\_A$  is invalid.

#### Stage 2:

1.  $CTXT\_A$  is invalid.

#### Stage 3:

1. This subroutine has been called from outside the process grid.

#### Stage 4:

1.  $m < 0$
2.  $n < 0$
3.  $M\_A < 0$  if  $m = 0$  or  $n = 0$ ;  $M\_A < 1$  otherwise
4.  $N\_A < 0$  if  $m = 0$  or  $n = 0$ ;  $N\_A < 1$  otherwise
5.  $MB\_A < 1$
6.  $NB\_A < 1$
7.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
8.  $CSRC\_A < 0$  or  $CSRC\_A \geq q$
9.  $ia < 1$
10.  $ja < 1$

#### Stage 5: If $m \neq 0$ and $n \neq 0$ :

1.  $ia > M\_A$
2.  $ja > N\_A$
3.  $ia+m-1 > M\_A$
4.  $ja+n-1 > N\_A$

In all cases:

1.  $MB\_A \neq NB\_A$

## PDGEBRD and PZGEBRD

### Stage 6:

1.  $\text{mod}(ia-1, MB\_A) \neq \text{mod}(ja-1, NB\_A)$
2.  $LLD\_A < \max(1, LOCp(M\_A))$
3.  $lwork \neq 0$ ,  $lwork \neq -1$ , and  $lwork < nb(mp0+nq0+1)+nq0$

where:

- $nb = MB\_A = NB\_A$
- $iroffa = \text{mod}(ia-1, nb)$
- $icoffa = \text{mod}(ja-1, nb)$
- $iarow = \text{mod}(RSRC\_A+(ia-1)/nb, nprow)$ .
- $iacol = \text{mod}(CSRC\_A+(ja-1)/nb, npc0l)$ .
- $mp0 = \text{NUMROC}(m+iroffa, nb, myrow, iarow, nprow)$
- $nq0 = \text{NUMROC}(n+icoffa, nb, mycol, iacol, npc0l)$

### Stage 7:

Each of the following global input arguments are checked to determine whether its value differs from the value specified on process  $P_{00}$ :

1.  $m$  differs.
2.  $n$  differs.
3.  $ia$  differs.
4.  $ja$  differs.
5.  $DTYPE\_A$  differs.
6.  $M\_A$  differs.
7.  $N\_A$  differs.
8.  $MB\_A$  differs.
9.  $NB\_A$  differs.
10.  $RSRC\_A$  differs.
11.  $CSRC\_A$  differs.

Also:

12.  $lwork = -1$  on a subset of processes.

## Examples

### Example 1

This example shows the reduction of a real general matrix of size 4 by 3 to bidiagonal form using a  $2 \times 2$  process grid.

**Note:** Because  $lwork = 0$ , PDGEBRD dynamically allocates the work area used by this subroutine.

### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      M   N   A   IA   JA   DESC_A   D   E   TAUQ   TAUP   WORK   LWORK   INFO
      |   |   |   |   |   |         |   |   |       |       |       |
CALL PDGEBRD( 4 , 3 , A , 1 , 1 , DESC_A , D , E , TAUQ , TAUP , WORK , 0 , INFO )
```

	DESC_A
DTYPE_	1
CTXT_	<i>icontxt</i> <sup>1</sup>

	DESC_A
M_	4
N_	3
MB_	2
NB_	2
RSRC_	0
CSRC_	0
LLD_	See below <sup>2</sup>

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:  

$$\text{LLD\_A} = \text{MAX}(1, \text{NUMROC}(\text{M\_A}, \text{MB\_A}, \text{MYROW}, \text{RSRC\_A}, \text{NPROW}))$$
In this example,  $\text{LLD\_A} = 2$  on all processes.

Global general matrix  $A$  of size  $4 \times 3$  with block sizes  $2 \times 2$ :

B,D	0	1
0	$\begin{bmatrix} 10.0 & 5.0 \\ 2.0 & 16.0 \end{bmatrix}$	$\begin{bmatrix} 9.0 \\ 10.0 \end{bmatrix}$
1	$\begin{bmatrix} 3.0 & 7.0 \\ 4.0 & 8.0 \end{bmatrix}$	$\begin{bmatrix} 21.0 \\ 12.0 \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0	1
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$

Local arrays for  $A$ :

p,q	0	1
0	$\begin{bmatrix} 10.0 & 5.0 \\ 2.0 & 16.0 \end{bmatrix}$	$\begin{bmatrix} 9.0 \\ 10.0 \end{bmatrix}$
1	$\begin{bmatrix} 3.0 & 7.0 \\ 4.0 & 8.0 \end{bmatrix}$	$\begin{bmatrix} 21.0 \\ 12.0 \end{bmatrix}$

**Output:**

Global general matrix  $A$  of size  $4 \times 3$  with block sizes  $2 \times 2$ :

B,D	0	1
0	$\begin{bmatrix} -11.36 & 22.80 \\ 0.09 & 23.32 \end{bmatrix}$	$\begin{bmatrix} 0.56 \\ 1.67 \end{bmatrix}$
1	$\begin{bmatrix} 0.14 & 0.46 \\ 0.19 & 0.22 \end{bmatrix}$	$\begin{bmatrix} -9.68 \\ 0.08 \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

## PDGEBRD and PZGEBRD

B,D	0	1
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$

Local arrays for  $A$ :

p,q	0	1
0	-11.36 22.80 0.09 23.32	0.56 1.67
1	0.14 0.46 0.19 0.22	-9.68 0.08

Global row vector  $D$  of length 3 with block size 2:

B,D	0	1
0	$\begin{bmatrix} -11.36 & 23.32 &   & -9.68 \end{bmatrix}$	

**Note:** A copy of  $D$  is distributed across each row of the process grid.

The following is the  $2 \times 2$  process grid:

B,D	0	1
	$P_{00}$	$P_{01}$
	$P_{10}$	$P_{11}$

Local arrays for  $D$ :

p,q	0	1
0	-11.36 23.32	-9.68
1	-11.36 23.32	-9.68

Global column vector  $E$  of length 2 with block size 2:

B,D	0
0	$\begin{bmatrix} 22.80 \\ 1.67 \end{bmatrix}$

**Note:** A copy of  $E$  is distributed across each column of the process grid.

The following is the  $2 \times 2$  process grid:

B,D		
0	$P_{00}$	$P_{01}$
	$P_{10}$	$P_{11}$

Local arrays for  $E$ :

p,q	0	1
0	22.80 1.67	22.80 1.67
1	.	.



Global row vector  $\tau_q$  of length 3 with block size 2:

$$\begin{array}{c} \text{B,D} \quad \quad \quad 0 \quad \quad \quad 1 \\ 0 \quad \left[ \begin{array}{cc|c} 1.88 & 1.59 & 1.99 \end{array} \right] \end{array}$$

**Note:** A copy of  $\tau_q$  is distributed across each row of the process grid.

The following is the  $2 \times 2$  process grid:

$$\begin{array}{c|c|c} \text{B,D} & 0 & 1 \\ \hline & P_{00} & P_{01} \\ \hline & P_{10} & P_{11} \end{array}$$

Local arrays for  $\tau_q$ :

$$\begin{array}{c|c|c} \text{p,q} & 0 & 1 \\ \hline 0 & 1.88 & 1.59 & 1.99 \\ \hline 1 & 1.88 & 1.59 & 1.99 \end{array}$$

Global column vector  $\tau_p$  of length 3 with block size 2:

$$\begin{array}{c} \text{B,D} \quad \quad 0 \\ 0 \quad \left[ \begin{array}{c} 1.52 \\ 0.00 \end{array} \right] \\ 1 \quad \left[ \begin{array}{c} 0.00 \end{array} \right] \end{array}$$

**Note:** A copy of  $\tau_p$  is distributed across each column of the process grid.

The following is the  $2 \times 2$  process grid:

$$\begin{array}{c|c|c} \text{B,D} & & \\ \hline 0 & P_{00} & P_{01} \\ \hline 1 & P_{10} & P_{11} \end{array}$$

Local arrays for  $\tau_p$ :

$$\begin{array}{c|c|c} \text{p,q} & 0 & 1 \\ \hline 0 & 1.52 & 1.52 \\ & 0.00 & 0.00 \\ \hline 1 & 0.00 & 0.00 \end{array}$$

The value of *info* is 0 on all processes.

## Example 2

This example shows the reduction of a complex general matrix of size 3 by 4 to bidiagonal form using a  $2 \times 2$  process grid.

**Note:** Because *lwork* = 0, PZGEBRD dynamically allocates the work area used by this subroutine.

## PDGEBRD and PZGEBRD

### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      M   N   A   IA   JA   DESC_A   D   E   TAUQ   TAUP   WORK   LWORK   INFO
      |   |   |   |   |   |         |   |   |       |       |       |
CALL PZGEBRD( 3 , 4 , A , 1 , 1 , DESC_A , D , E , TAUQ , TAUP , WORK , 0 , INFO )
```

	DESC_A
DTYPE_A	1
CTXT_	<i>icontxt</i> <sup>1</sup>
M_	3
N_	4
MB_	2
NB_	2
RSRC_	0
CSRC_	0
LLD_	See below <sup>2</sup>

### Notes:

- icontxt* is the output of the BLACS\_GRIDINIT call.
- Each process should set the LLD\_ as follows:  
 $LLD\_A = \text{MAX}(1, \text{NUMROC}(M\_A, MB\_A, MYROW, RSRC\_A, NPROW))$   
 In this example,  $LLD\_A = 2$  on  $P_{00}$  and  $P_{01}$  and  $LLD\_A = 1$  on  $P_{10}$  and  $P_{11}$ .

Global complex general matrix  $A$  of size  $3 \times 4$  with block sizes  $2 \times 2$ :

$$\begin{array}{c} \text{B,D} \end{array} \quad \begin{array}{cc} 0 & 1 \end{array} \quad \left[ \begin{array}{cc|cc} 0 & \begin{pmatrix} 1.00 & 1.00 \\ -1.00 & 1.00 \end{pmatrix} & \begin{pmatrix} 1.00 & -1.00 \\ -1.00 & 0.00 \end{pmatrix} & \begin{pmatrix} 2.00 & 0.00 \\ 0.00 & -2.00 \end{pmatrix} & \begin{pmatrix} 0.00 & -2.00 \\ 2.00 & 2.00 \end{pmatrix} \\ 1 & \begin{pmatrix} 0.00 & -1.00 \\ -1.00 & -1.00 \end{pmatrix} & \begin{pmatrix} -1.00 & -1.00 \end{pmatrix} & \begin{pmatrix} -2.00 & 2.00 \end{pmatrix} & \begin{pmatrix} 2.00 & -2.00 \end{pmatrix} \end{array} \right]$$

The following is the  $2 \times 2$  process grid:

B,D	0	1
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$

Local arrays for  $A$ :

p,q	0	1
0	$\begin{pmatrix} 1.00 & 1.00 \\ -1.00 & 1.00 \end{pmatrix}$	$\begin{pmatrix} 1.00 & -1.00 \\ -1.00 & 0.00 \end{pmatrix}$
1	$\begin{pmatrix} 0.00 & -1.00 \\ -1.00 & -1.00 \end{pmatrix}$	$\begin{pmatrix} 2.00 & 0.00 \\ 0.00 & -2.00 \end{pmatrix}$

### Output:

Global general matrix  $A$  of size  $3 \times 4$  with block sizes  $2 \times 2$ :

$$\begin{array}{c} B,D \end{array} \begin{array}{cc} 0 & 1 \end{array} \left[ \begin{array}{cc|cc} 0 & \begin{pmatrix} -3.46, & 0.00 \\ -2.08, & 0.00 \end{pmatrix} & \begin{pmatrix} 0.17, & -0.26 \\ 4.51, & 0.00 \end{pmatrix} & \begin{pmatrix} 0.43, & -0.10 \\ 0.10, & -0.59 \end{pmatrix} & \begin{pmatrix} -0.10, & -0.43 \\ 0.33, & 0.52 \end{pmatrix} \\ 1 & \begin{pmatrix} 0.11, & -0.40 \end{pmatrix} & \begin{pmatrix} -1.73, & 0.00 \end{pmatrix} & \begin{pmatrix} 2.51, & 0.00 \end{pmatrix} & \begin{pmatrix} 0.01, & 0.11 \end{pmatrix} \end{array} \right]$$

The following is the  $2 \times 2$  process grid:

$$\begin{array}{c} B,D \end{array} \left| \begin{array}{c} 0 \\ 1 \end{array} \right| \begin{array}{c} 1 \\ 1 \end{array} \begin{array}{c} P_{00} \\ P_{10} \end{array} \left| \begin{array}{c} P_{01} \\ P_{11} \end{array} \right|$$

Local arrays for  $A$ :

$$\begin{array}{c} p,q \end{array} \left| \begin{array}{cc} 0 & 1 \end{array} \right| \begin{array}{cc} 0 & 1 \end{array} \left[ \begin{array}{cc|cc} 0 & \begin{pmatrix} -3.46, & 0.00 \\ -2.08, & 0.00 \end{pmatrix} & \begin{pmatrix} 0.17, & -0.26 \\ 4.51, & 0.00 \end{pmatrix} & \begin{pmatrix} 0.43, & -0.10 \\ 0.10, & -0.59 \end{pmatrix} & \begin{pmatrix} -0.10, & -0.43 \\ 0.33, & 0.52 \end{pmatrix} \\ 1 & \begin{pmatrix} 0.11, & -0.40 \end{pmatrix} & \begin{pmatrix} -1.73, & 0.00 \end{pmatrix} & \begin{pmatrix} 2.51, & 0.00 \end{pmatrix} & \begin{pmatrix} 0.01, & 0.11 \end{pmatrix} \end{array} \right]$$

Global column vector  $D$  of length 3 with block size 2:

$$\begin{array}{c} B,D \end{array} \begin{array}{c} 0 \end{array} \left[ \begin{array}{c} -3.46 \\ 4.51 \\ 2.51 \end{array} \right]$$

**Note:** A copy of  $D$  is distributed across each column of the process grid.

The following is the  $2 \times 2$  process grid:

$$\begin{array}{c} B,D \end{array} \left| \begin{array}{c} 0 \\ 1 \end{array} \right| \begin{array}{c} P_{00} \\ P_{10} \end{array} \left| \begin{array}{c} P_{01} \\ P_{11} \end{array} \right|$$

Local arrays for  $D$ :

$$\begin{array}{c} p,q \end{array} \left| \begin{array}{c} 0 \\ 1 \end{array} \right| \begin{array}{c} 0 \\ 1 \end{array} \left[ \begin{array}{c|c} 0 & -3.46 \\ 0 & 4.51 \\ 1 & 2.51 \end{array} \right]$$

Global row vector  $E$  of length 2 with block size 2:

$$\begin{array}{c} B,D \end{array} \begin{array}{c} 0 \end{array} \left[ \begin{array}{cc} -2.08 & -1.73 \end{array} \right]$$

**Note:** A copy of  $E$  is distributed across each row of the process grid.

The following is the  $2 \times 2$  process grid:

B,D	0	
	P <sub>00</sub>	P <sub>01</sub>
	P <sub>10</sub>	P <sub>11</sub>

Local arrays for  $E$ :

p,q	0	1
0	-2.08 -1.73	.
1	-2.08 -1.73	.

Global row vector  $\tau_q$  of length 3 with block size 2:

B,D	0	1
0	[ (1.69, -0.14) (1.91, 0.41)   (0.00, 0.00) ]	

**Note:** A copy of  $\tau_q$  is distributed across each row of the process grid.

The following is the  $2 \times 2$  process grid:

B,D	0	1
	P <sub>00</sub>	P <sub>01</sub>
	P <sub>10</sub>	P <sub>11</sub>

Local arrays for  $\tau_q$ :

p,q	0	1
0	(1.69, -0.14) (1.91, 0.41)	(0.00, 0.00)
1	(1.69, -0.14) (1.91, 0.41)	(0.00, 0.00)

Global column vector  $\tau_p$  of length 3 with block size 2:

B,D	0
0	[ (1.29, -0.29) (1.03, 0.36) ]
1	

**Note:** A copy of  $\tau_p$  is distributed across each column of the process grid.

The following is the  $2 \times 2$  process grid:

B,D		
0	P <sub>00</sub>	P <sub>01</sub>
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for  $\tau_p$ :

p,q	0	1
0	(1.29, -0.29) (1.03, 0.36)	(1.29, -0.29) (1.03, 0.36)
1	(1.88, -0.43)	(1.88, -0.43)

The value of *info* is 0 on all processes.

## PDGESVD and PZGESVD — Singular Value Decomposition of a General Matrix

### Purpose

These subroutines compute the singular values and, optionally, the left and/or right singular vectors of a real or complex general matrix  $A$ , where  $A$  represents the global real or complex submatrix  $A_{ia:ia+m-1, ja:ja+n-1}$ .

See references [13], [22], [26], [37], and [49].

Table 122. Data Types

$A$ , $U$ , $VT$ , $work$	$s$ , $rwork$	Subroutine
Long-precision real	Long-precision real	PDGESVD
Long-precision complex	Long-precision real	PZGESVD

### Syntax

<b>Fortran</b>	CALL PDGESVD ( <i>jobu</i> , <i>jobvt</i> , <i>m</i> , <i>n</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>s</i> , <i>u</i> , <i>iu</i> , <i>ju</i> , <i>desc_u</i> , <i>vt</i> , <i>ivt</i> , <i>jvt</i> , <i>desc_vt</i> , <i>work</i> , <i>lwork</i> , <i>info</i> )  CALL PZGESVD ( <i>jobu</i> , <i>jobvt</i> , <i>m</i> , <i>n</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>s</i> , <i>u</i> , <i>iu</i> , <i>ju</i> , <i>desc_u</i> , <i>vt</i> , <i>ivt</i> , <i>jvt</i> , <i>desc_vt</i> , <i>work</i> , <i>lwork</i> , <i>rwork</i> , <i>lrwork</i> , <i>info</i> )
<b>C and C++</b>	pdgesvd ( <i>jobu</i> , <i>jobvt</i> , <i>m</i> , <i>n</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>s</i> , <i>u</i> , <i>iu</i> , <i>ju</i> , <i>desc_u</i> , <i>vt</i> , <i>ivt</i> , <i>jvt</i> , <i>desc_vt</i> , <i>work</i> , <i>lwork</i> , <i>info</i> );  pzgesvd ( <i>jobu</i> , <i>jobvt</i> , <i>m</i> , <i>n</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>s</i> , <i>u</i> , <i>iu</i> , <i>ju</i> , <i>desc_u</i> , <i>vt</i> , <i>ivt</i> , <i>jvt</i> , <i>desc_vt</i> , <i>work</i> , <i>lwork</i> , <i>rwork</i> , <i>lrwork</i> , <i>info</i> );

### On Entry

*jobu* specifies the type of computation, where:

- If *jobu* = 'N', the left singular vectors are not computed.
- If *jobu* = 'V', the left singular vectors are computed.

Scope: **global**

Specified as: a single character; *jobu* = 'N' or 'V'

*jobvt* specifies the type of computation, where:

- If *jobvt* = 'N', the right singular vectors are not computed.
- If *jobvt* = 'V', the right singular vectors are computed.

Scope: **global**

Specified as: a single character; *jobvt* = 'N' or 'V'

*m* is the number of rows of submatrix  $A$  used in the computation.

Scope: **global**

Specified as: a fullword integer;  $m \geq 0$

*n* is the number of columns of submatrix  $A$  used in the computation.

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

*a* is the local part of the global general matrix  $A$ . This identifies the **first**

**element** of the local array  $A$ . This subroutine computes the location of the first element of the local subarray used, based on  $ia$ ,  $ja$ ,  $desc\_a$ ,  $p$ ,  $q$ ,  $myrow$ , and  $mycol$ ; therefore, the leading  $LOCp(ia+m-1)$  by  $LOCq(ja+n-1)$  part of the local array  $A$  must contain the local pieces of the leading  $ia+m-1$  by  $ja+n-1$  part of the global matrix.

Scope: **local**

Specified as: an  $LLD\_A$  by (at least)  $LOCq(N\_A)$  array, containing numbers of the data type indicated in Table 122 on page 780. Details about the square block-cyclic data distribution of global matrix  $A$  are stored in  $desc\_a$ .

$ia$  is the row index of the global matrix  $A$ , identifying the first row of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $ia = 1$ .

$ja$  is the column index of the global matrix  $A$ , identifying the first column of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $ja = 1$ .

$desc\_a$  is the array descriptor for global matrix  $A$ , described in the following table:

$desc\_a$	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $m = 0$ or $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	If $m = 0$ or $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_A < p$	Global
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < q$	Global
9	LLD_A	The leading dimension of the local array	$LLD\_A \geq \max(1, LOCp(M\_A))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

$iu$  is the row index of the global matrix  $U$ , identifying the first row of the submatrix  $U$ .

Scope: **global**

Specified as: a fullword integer;  $iu = 1$ .

## PDGESVD and PZGESVD

$ju$  is the column index of the global matrix  $U$ , identifying the first column of the submatrix  $U$ .

Scope: **global**

Specified as: a fullword integer;  $ju = 1$ .

$desc\_u$  is the array descriptor for global matrix  $U$ , described in the following table:

$desc\_u$	Name	Description	Limits	Scope
1	DTYPE_U	Descriptor type	DTYPE_U=1	Global
2	CTXT_U	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_U	Number of rows in the global matrix	If $m = 0$ or $\min(m, n) = 0$ : $M\_U \geq 0$  Otherwise: $M\_U \geq 1$	Global
4	N_U	Number of columns in the global matrix	If $m = 0$ or $\min(m, n) = 0$ : $N\_U \geq 0$  Otherwise: $N\_U \geq 1$	Global
5	MB_U	Row block size	$MB\_U \geq 1$	Global
6	NB_U	Column block size	$NB\_U \geq 1$	Global
7	RSRC_U	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_U < p$	Global
8	CSRC_U	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_U < q$	Global
9	LLD_U	The leading dimension of the local array	$LLD\_U \geq \max(1, LOCp(M\_U))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

$ivt$  is the row index of the global matrix  $VT$ , identifying the first row of the submatrix  $VT$ .

Scope: **global**

Specified as: a fullword integer;  $ivt = 1$ .

$jvt$  is the column index of the global matrix  $VT$ , identifying the first column of the submatrix  $VT$ .

Scope: **global**

Specified as: a fullword integer;  $jvt = 1$ .

$desc\_vt$  is the array descriptor for global matrix  $VT$ , described in the following table:

$desc\_vt$	Name	Description	Limits	Scope
1	DTYPE_VT	Descriptor type	DTYPE_VT=1	Global
2	CTXT_VT	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global



<i>desc_vt</i>	Name	Description	Limits	Scope
3	M_VT	Number of rows in the global matrix	If $\min(m, n) = 0$ or $n = 0$ : $M\_VT \geq 0$  Otherwise: $M\_VT \geq 1$	Global
4	N_VT	Number of columns in the global matrix	If $\min(m, n) = 0$ or $n = 0$ : $N\_VT \geq 0$  Otherwise: $N\_VT \geq 1$	Global
5	MB_VT	Row block size	$MB\_VT \geq 1$	Global
6	NB_VT	Column block size	$NB\_VT \geq 1$	Global
7	RSRC_VT	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_VT < p$	Global
8	CSRC_VT	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_VT < q$	Global
9	LLD_VT	The leading dimension of the local array	$LLD\_VT \geq \max(1, LOCp(M\_VT))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

*work* has the following meaning:

If  $lwork = 0$ , *work* is ignored.

If  $lwork \neq 0$ , *work* is the work area used by this subroutine, where:

- If  $lwork \neq -1$ , its size is (at least) of length *lwork*.
- If  $lwork = -1$ , its size is (at least) of length 1.

Scope: **local**

Specified as: an area of storage containing numbers of data type indicated in Table 122 on page 780.

*lwork* is the number of elements in array WORK.

Scope:

- If  $lwork \geq 0$ , *lwork* is **local**
- If  $lwork = -1$ , *lwork* is **global**

Specified as: a fullword integer; where:

- If  $lwork = 0$ , PDGESVD and PZGESVD dynamically allocate the work area used by these subroutines. The work area is deallocated before control is returned to the calling program. This option is an extension to the ScaLAPACK standard.
- If  $lwork = -1$ , PDGESVD and PZGESVD perform a work area query and return the minimum size of *work* in *work*<sub>1</sub>. No computation is performed and the subroutines return after error checking is complete.
- Otherwise, use the following rules to determine the value to specify:

**For PDGESVD**,  $lwork \geq 2 + 6 \cdot \max(m, n) + \max(watobd, wbdtsvd)$

where:

- $nb = MB\_A = NB\_A$
- $mp = NUMROC(m, nb, myrow, 0, nprow)$

- $nq = \text{NUMROC}(n, nb, mycol, 0, npcol)$
- $mp0 = \text{NUMROC}(m, nb, 0, 0, nprow)$
- $nq0 = \text{NUMROC}(n, nb, 0, 0, npcol)$
- $watobd = \max(wpdlange, wpdgebrd, wpdlared2d, wpdlared1d)$
- $wpdlange = nq$
- $wpdgebrd = nb(mp + nq + 1) + nq$
- $wpdlared1d = nq0$
- $wpdlared2d = nq0$
- $wbdtosvd = \min(m, n)*(nru + ncvt) + \max(wdbdsqr, wpdormbrqln, wpdormbrprt)$
- If  $jobu = 'V'$  or  $jobvt = 'V'$ , then:
- $wdbdsqr = \max(1, 4*\min(m, n)-4)$
- Otherwise:
- $wdbdsqr = \max(1, 2*\min(m, n))$
- If  $jobu = 'V'$ , then:
- $myrowc = myrow*npcol + mycol$
- $nru = \text{NUMROC}(\min(m, n), 1, myrowc, 0, (nprow)(npcol))$
- $sizeq = \text{NUMROC}(\min(m, n), nb, mycol, 0, npcol)$
- $wpdormbrqln = \max(nb*(nb - 1) / 2, (sizeq + mp)*nb) + nb*nb$
- Otherwise:
- $nru = 0$
- $wpdormbrqln = 0$
- If  $jobvt = 'V'$ , then:
- $mycolr = myrow*npcol + mycol$
- $ncvt = \text{NUMROC}(\min(m, n), 1, mycolr, 0, (nprow)(npcol))$
- $sizep = \text{NUMROC}(\min(m, n), nb, myrow, 0, nprow)$
- $wpdormbrprt = \max(nb*(nb - 1) / 2, (sizep + nq)*nb) + nb*nb$
- Otherwise:
- $ncvt = 0$
- $wpdormbrprt = 0$

**For PZGESVD,  $lwork \geq 1 + 2*\max(m, n) + \max(watobdc, wbdtosvdc)$**

where:

- $nb = MB\_A = NB\_A$
- $mp = \text{NUMROC}(m, nb, myrow, 0, nprow)$
- $nq = \text{NUMROC}(n, nb, mycol, 0, npcol)$
- $mp0 = \text{NUMROC}(m, nb, 0, 0, nprow)$
- $nq0 = \text{NUMROC}(n, nb, 0, 0, npcol)$
- $watobdc = wpzgebrd$
- $wpzgebrd = nb(mp + nq + 1) + nq$
- $wbdtosvdc = \min(m, n)*(nru + ncvt + \max(wpzunmbrqln, wpzunmbrprt))$
- If  $jobu = 'V'$ , then:
- $myrowc = myrow*npcol + mycol$
- $nru = \text{NUMROC}(\min(m, n), 1, myrowc, 0, (nprow)(npcol))$
- $sizeq = \text{NUMROC}(\min(m, n), nb, mycol, 0, npcol)$
- $wpzunmbrqln = \max(nb*(nb - 1) / 2, (sizeq + mp)*nb) + nb*nb$
- Otherwise:
- $nru = 0$
- $wpzunmbrqln = 0$
- If  $jobvt = 'V'$ , then:
- $mycolr = myrow*npcol + mycol$
- $ncvt = \text{NUMROC}(\min(m, n), 1, mycolr, 0, (nprow)(npcol))$
- $sizep = \text{NUMROC}(\min(m, n), nb, myrow, 0, nprow)$
- $wpzunmbrprt = \max(nb*(nb - 1) / 2, (sizep + nq)*nb) + nb*nb$
- Otherwise:

- $ncvt = 0$
- $wpzunmbrprt = 0$

*rwork* has the following meaning:

If  $lrwork = 0$ , *rwork* is ignored.

If  $lrwork \neq 0$ , *rwork* is the work area used by this subroutine, where:

- If  $lrwork \neq -1$ , its size is (at least) of length *lrwork*.
- If  $lrwork = -1$ , its size is (at least) of length 1.

Scope: **local**

Specified as: an area of storage containing numbers of data type indicated in Table 122 on page 780.

*lrwork* is the number of elements in array WORK.

Scope:

- If  $lrwork \geq 0$ , *lrwork* is **local**
- If  $lrwork = -1$ , *lrwork* is **global**

Specified as: a fullword integer; where:

- If  $lrwork = 0$ , PZGESVD dynamically allocates the work area used by this subroutine. The work area is deallocated before control is returned to the calling program. This option is an extension to the ScaLAPACK standard.
- If  $lrwork = -1$ , PZGESVD performs a work area query and return the minimum size of *rwork* in *rwork*<sub>1</sub>. No computation is performed and the subroutines return after error checking is complete.
- Otherwise, use the following rules to determine the value to specify:  
 $lrwork \geq 1 + 4 \cdot \max(m, n) + \max(watobdr, wbdtsvdr)$

where:

- $nb = MB\_A = NB\_A$
- $mp = \text{NUMROC}(m, nb, myrow, 0, nprow)$
- $nq = \text{NUMROC}(n, nb, mycol, 0, npcol)$
- $mp0 = \text{NUMROC}(m, nb, 0, 0, nprow)$
- $nq0 = \text{NUMROC}(n, nb, 0, 0, npcol)$
- $watobdr = \max(wpzlange, wplared2d, wplared1d)$
- $wplange = nq$
- $wplared1d = nq0$
- $wplared2d = mp0$
- $wbdtsvdr = wzbdsqr$
- If  $jobu = 'V'$  or  $jobvt = 'V'$ , then:
- $wzbdsqr = \max(1, 4 \cdot \min(m, n) - 4)$
- Otherwise:
- $wzbdsqr = \max(1, 2 \cdot \min(m, n))$

**Note:** Dynamically allocating the workspace is a Parallel ESSL extension to the ScaLAPACK standard.

*info* See On Return.

## On Return

*a* is the updated local part of the global general matrix A. Matrix A is overwritten; that is, the original input is not preserved.

Scope: **local**

Returned as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 122 on page 780. Details about the square block-cyclic data distribution of global matrix *A* are stored in *desc\_a*.

*s* On normal exit (see output argument *info* on page 787), it is the vector *s*, containing the singular values in descending order in the first min(*m*,*n*) elements of *s*.

Scope: **global**

Returned as: a one-dimensional array of at least length min(*m*, *n*), containing numbers of the data type indicated in Table 122 on page 780.

*u* is the updated local part of the global matrix *U*, where:

- If *jobu* = 'N', then *u* is ignored.
- If *jobu* = 'V', then this is the updated local part of the global matrix *U*, where columns *ju* to *ju*+min(*m*,*n*)-1 of the global matrix *U* contain the left singular vectors of the global matrix *A*.

This identifies the first element of the local array *U*. This subroutine computes the location of the first element of the local subarray used, based on *iu*, *ju*, *desc\_u*, *p*, *q*, *myrow*, and *mycol*; therefore, the LOCp(*iu*+*m*-1) by LOCq(*ju*+min(*m*,*n*)-1) part of the local array *U* contains the local pieces of the leading *iu*+*m*-1 by *ju*+min(*m*,*n*)-1 part of the global matrix *U*.

Scope: **local**

Returned as: an LLD\_U by (at least) LOCq(N\_U) array, containing numbers of the data type indicated in Table 122 on page 780.

*vt* is the local part of the global matrix *VT*, where:

- If *jobvt* = 'N', then *vt* is ignored.
- If *jobvt* = 'V', then this is the updated local part of the global matrix *VT*, where rows *ivt* to *ivt*+min(*m*,*n*)-1 of the global matrix *VT* contain the right singular vectors of the global matrix *A*.

This identifies the first element of the local array *VT*. This subroutine computes the location of the first element of the local subarray used, based on *ivt*, *jvt*, *desc\_vt*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading LOCp(*ivt*+min(*m*, *n*)-1) by LOCq(*jvt*+*n*-1) part of the local array *VT* contains the local pieces of the leading *ivt*+min(*m*, *n*)-1 by *jvt*+*n*-1 part of the global matrix *VT*.

Scope: **local**

Returned as: an LLD\_VT by (at least) LOCq(N\_VT) array, containing numbers of the data type indicated in Table 122 on page 780.

*work* is the work area used by this subroutine if *lwork* ≠ 0, where:

If *lwork* ≠ 0 and *lwork* ≠ -1, its size is (at least) of length *lwork*.

If *lwork* = -1, its size is (at least) of length 1.

Scope: **local**

Returned as: an area of storage, where:

If *lwork* ≥ 1 or *lwork* = -1, then *work*<sub>1</sub> is set to the minimum *lwork* value and contains numbers of the data type indicated in Table 122 on page 780. Except for *work*<sub>1</sub>, the contents of *work* are overwritten on return.

*rwork* is the work area used by this subroutine if *lrwork* ≠ 0, where:

If *lrwork* ≠ 0 and *lrwork* ≠ -1, its size is (at least) of length *lrwork*.

If  $lrwork = -1$ , its size is (at least) of length 1.

Scope: **local**

Returned as: an area of storage, where:

If  $lrwork \geq 1$  or  $lrwork = -1$ , then  $rwork_1$  is set to the minimum  $lrwork$  value and contains numbers of the data type indicated in Table 122 on page 780. Except for  $rwork_1$ , the contents of  $rwork$  are overwritten on return.

*info* has the following meaning:

If  $info = 0$ , then no computational errors occurred. This indicates a normal exit.

If  $info \geq 0$ , then one of the following computational errors occurred:

- If  $info \leq \min(m,n)$ , the singular values of general matrix  $A$  failed to converge.  $info$  is set equal to  $i$ , the number of elements of the superdiagonal of an intermediate bidiagonal form that failed to converge to zero.
- If  $info = \min(m,n)+1$ , the singular values are not the same on all processes.

Scope: **global**

Returned as: a fullword integer;  $info \geq 0$ .

## Notes and Coding Rules

1. In your C program, argument *info* must be passed by reference.
2.  $A$ ,  $s$ ,  $U$ ,  $VT$ ,  $work$ , and  $rwork$  must have no common elements; otherwise, results are unpredictable.
3. The NUMROC utility subroutine can be used to determine the values of LOCp(M\_) and LOCq(N\_) used in the argument descriptions above. For details, see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28 and “NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process” on page 868.
4. The global general matrix  $A$  must be distributed using a square block-cyclic distribution; that is,  $MB\_A = NB\_A$ .

5. In all cases, follow these rules:

- $ia = ja = 1$
- $RSRC\_A = CSRC\_A = 0$

If  $jobu = 'V'$ , then also follow these rules:

- $iu = ju = 1$
- $M\_A = M\_U$
- $MB\_A = MB\_U$
- $NB\_A = NB\_U$
- $RSRC\_U = CSRC\_U = 0$
- $CTXT\_A = CTXT\_U$

If  $jobvt = 'V'$ , then also follow these rules:

- $ivt = jvt = 1$
- $N\_A = N\_VT$
- $MB\_A = MB\_VT$
- $NB\_A = NB\_VT$
- $RSRC\_VT = CSRC\_VT = 0$
- $CTXT\_A = CTXT\_VT$

6. For suggested block sizes, see “Coding Tips for Optimizing Parallel Performance” on page 77.
7. If  $lwork = -1$  on any process, it must equal  $-1$  on all processes. That is, if a subset of the processes specifies  $-1$  for the work area size, they must all specify  $-1$ .
8. If  $lrwork = -1$  on any process, it must equal  $-1$  on all processes. That is, if a subset of the processes specifies  $-1$  for the work area size, they must all specify  $-1$ .

## Error Conditions

### Computational Errors

**Note:** For more details, see output argument *info* on page 787.

- The singular values failed to converge.
- The singular values are not the same on all processes.

### Resource Errors

1.  $lwork = 0$  and unable to allocate work space.
2.  $lrwork = 0$  and unable to allocate work space.

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. DTYPE\_A is invalid.
2. DTYPE\_U is invalid and  $jobu = 'V'$ .
3. DTYPE\_VT is invalid and  $jobvt = 'V'$ .

#### Stage 2:

1. CTEXT\_A is invalid.

#### Stage 3:

1. This subroutine has been called from outside the process grid.

#### Stage 4:

1.  $jobu \neq 'N'$  or  $'V'$ .
2.  $jobvt \neq 'N'$  or  $'V'$ .
3.  $m < 0$
4.  $n < 0$
5.  $M\_A < 0$  if  $m = 0$  or  $n = 0$ ;  $M\_A < 1$  otherwise
6.  $N\_A < 0$  if  $m = 0$  or  $n = 0$ ;  $N\_A < 1$  otherwise
7.  $MB\_A < 1$
8.  $NB\_A < 1$
9.  $RSRC\_A \neq 0$
10.  $CSRC\_A \neq 0$
11.  $ia \neq 1$
12.  $ja \neq 1$

If  $jobu = 'V'$ :

13.  $M\_U < 0$  if  $m = 0$  or  $\min(m,n) = 0$ ;  $M\_U < 1$  otherwise
14.  $N\_U < 0$  if  $m = 0$  or  $\min(m,n) = 0$ ;  $N\_U < 1$  otherwise
15.  $MB\_U < 1$
16.  $NB\_U < 1$
17.  $RSRC\_U \neq 0$
18.  $CSRC\_U \neq 0$

19.  $iu \neq 1$
20.  $ju \neq 1$
21.  $CTXT\_A \neq CTXT\_U$

If  $jobvt = 'V'$ :

22.  $M\_VT < 0$  if  $\min(m,n) = 0$  or  $n = 0$ ;  $M\_VT < 1$  otherwise
23.  $N\_VT < 0$  if  $\min(m,n) = 0$  or  $n = 0$ ;  $N\_VT < 1$  otherwise
24.  $MB\_VT < 1$
25.  $NB\_VT < 1$
26.  $RSRC\_VT \neq 0$
27.  $CSRC\_VT \neq 0$
28.  $ivt \neq 1$
29.  $jvt \neq 1$
30.  $CTXT\_A \neq CTXT\_VT$

**Stage 5:** If  $m \neq 0$  and  $n \neq 0$ :

1.  $ia > M\_A$
2.  $ja > N\_A$
3.  $ia+m-1 > M\_A$
4.  $ja+n-1 > N\_A$

If  $jobu = 'V'$  and ( $m \neq 0$  and  $\min(m,n) \neq 0$ ):

5.  $iu > M\_U$
6.  $ju > N\_U$
7.  $iu+m-1 > M\_U$
8.  $ju+m-1 > N\_U$  and  $m \leq n$ ;  $ju+n-1 > N\_U$  and  $m > n$

If  $jobvt = 'V'$  and ( $\min(m,n) \neq 0$  and  $n \neq 0$ ):

9.  $ivt > M\_VT$
10.  $jvt > N\_VT$
11.  $ivt+m-1 > M\_VT$  and  $m \leq n$ ;  $ivt+n-1 > M\_VT$  and  $m > n$
12.  $jvt+n-1 > N\_VT$

In all cases:

13.  $MB\_A \neq NB\_A$

If  $jobu = 'V'$ :

14.  $M\_A \neq M\_U$
15.  $MB\_A \neq MB\_U$
16.  $NB\_A \neq NB\_U$

If  $jobvt = 'V'$ :

17.  $N\_A \neq N\_VT$
18.  $MB\_A \neq MB\_VT$
19.  $NB\_A \neq NB\_VT$

**Stage 6:**

1.  $LLD\_A < \max(1, LOCp(M\_A))$
2.  $lwork \neq 0$ ,  $lwork \neq -1$ , and  $lwork < (\text{minimum value})$  (For the minimum value, see the *lwork* argument description.)
3.  $lrwork \neq 0$ ,  $lrwork \neq -1$ , and  $lrwork < (\text{minimum value})$  (For the minimum value, see the *lrwork* argument description.)

If *jobu* = 'V':

4. LLD\_U = max(1, LOCp(M\_U))

If *jobvt* = 'V':

5. LLD\_VT = max(1, LOCp(M\_VT))

**Stage 7:** Each of the following global input arguments are checked to determine whether its value differs from the value specified on process P<sub>00</sub>:

1. *jobu* differs.
2. *jobvt* differs.
3. *m* differs.
4. *n* differs.
5. *ia* differs.
6. *ja* differs.
7. DTYPE\_A differs.
8. M\_A differs.
9. N\_A differs.
10. MB\_A differs.
11. NB\_A differs.
12. RSRC\_A differs.
13. CSRC\_A differs.
14. *lwork* = -1 on a subset of processes.
15. *lrwork* = -1 on a subset of processes.

**Stage 8:**

If *jobu* = 'V':

1. *iu* differs.
2. *ju* differs.
3. DTYPE\_U differs.
4. M\_U differs.
5. N\_U differs.
6. MB\_U differs.
7. NB\_U differs.
8. RSRC\_U differs.
9. CSRC\_U differs.

If *jobvt* = 'V':

10. *ivt* differs.
11. *jvt* differs.
12. DTYPE\_VT differs.
13. M\_VT differs.
14. N\_VT differs.
15. MB\_VT differs.
16. NB\_VT differs.
17. RSRC\_VT differs.
18. CSRC\_VT differs.

## Examples

### Example 1

This example computes the SVD of a real general matrix of size 4 by 3 using a 2 × 2 process grid. The input matrix *A* is the same as input matrix *A* in the PDGEBRD "Example 1" on page 772.



**Note:** Because  $lwork = 0$ , PDGESVD dynamically allocates the work area used by this subroutine.

### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      JOBU  JOBVT  M    N    A  IA  JA  DESC_A  S  U  IU  JU  DESC_U  VT  IVT  JVT  DESC_VT  WORK  LWORK  INFO
      |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
CALL PDGESVD( 'V', 'V', 4, 3, A, 1, 1, DESC_A, S, U, 1, 1, DESC_U, VT, 1, 1, DESC_VT, WORK, 0, INFO )
```

	DESC_A	DESC_U	DESC_VT
DTYPE_	1	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	4	4	3
N_	3	3	3
MB_	2	2	2
NB_	2	2	2
RSRC_	0	0	0
CSRC_	0	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>	See below <sup>2</sup>

### Notes:

- icontxt* is the output of the BLACS\_GRIDINIT call.
- Each process should set the LLD\_ as follows:
 

```
LLD_A = MAX(1, NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_U = MAX(1, NUMROC(M_U, MB_U, MYROW, RSRC_U, NPROW))
LLD_VT = MAX(1, NUMROC(M_VT, MB_VT, MYROW, RSRC_VT, NPROW))
```

In this example:

  - LLD\_A = 2 on all processes.
  - LLD\_U = 2 on all processes.
  - LLD\_VT = 2 on P<sub>00</sub> and P<sub>01</sub> and LLD\_VT = 1 on P<sub>10</sub> and P<sub>11</sub>

Global general matrix  $A$  of size  $4 \times 3$  with block sizes  $2 \times 2$ :

$$B, D \quad \begin{array}{cc} & 0 & 1 \\ \begin{array}{c} 0 \\ 1 \end{array} & \left[ \begin{array}{cc|cc} 10.0 & 5.0 & & 9.0 \\ 2.0 & 16.0 & & 10.0 \\ \hline 3.0 & 7.0 & & 21.0 \\ 4.0 & 8.0 & & 12.0 \end{array} \right] \end{array}$$

The following is the  $2 \times 2$  process grid:

$$B, D \quad \begin{array}{c|c|c} & 0 & 1 \\ \hline 0 & P_{00} & P_{01} \\ \hline 1 & P_{10} & P_{11} \end{array}$$

Local arrays for  $A$ :

## PDGESVD and PZGESVD

p,q	0	1
0	10.0 5.0 2.0 16.0	9.0 10.0
1	3.0 7.0 4.0 8.0	21.0 12.0

### Output:

Global vector  $s$  of length 3:

- $s = (33.64, 9.89, 7.70)$

Global general matrix  $U$  of size  $4 \times 3$  with block sizes  $2 \times 2$ :

B,D	0	1
0	$\begin{bmatrix} -0.37 & 0.28 \\ -0.51 & -0.82 \end{bmatrix}$	$\begin{bmatrix} 0.86 \\ -0.02 \end{bmatrix}$
1	$\begin{bmatrix} -0.64 & 0.50 \\ -0.44 & 0.00 \end{bmatrix}$	$\begin{bmatrix} -0.51 \\ 0.03 \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0	1
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$

Local arrays for  $U$ :

p,q	0	1
0	$\begin{bmatrix} -0.37 & 0.28 \\ -0.51 & -0.82 \end{bmatrix}$	$\begin{bmatrix} 0.86 \\ -0.02 \end{bmatrix}$
1	$\begin{bmatrix} -0.64 & 0.50 \\ -0.44 & 0.00 \end{bmatrix}$	$\begin{bmatrix} -0.51 \\ 0.03 \end{bmatrix}$

Global general matrix  $VT$  of size  $3 \times 3$  with block sizes  $2 \times 2$ :

B,D	0	1
0	$\begin{bmatrix} -0.25 & -0.54 \\ 0.26 & -0.84 \end{bmatrix}$	$\begin{bmatrix} -0.81 \\ 0.48 \end{bmatrix}$
1	$\begin{bmatrix} 0.93 & 0.09 \end{bmatrix}$	$\begin{bmatrix} -0.35 \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0	1
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$

Local arrays for  $VT$ :

p,q	0		1
0	-0.25	-0.54	-0.81
	0.26	-0.84	0.48
1	0.93	0.09	-0.35

The value of *info* is 0 on all processes.

## Example 2

This example computes the SVD of a complex general matrix of size 3 by 4 using a  $2 \times 2$  process grid. The input matrix *A* is the same as input matrix *A* in the PZGEBRD “Example 2” on page 775.

**Note:** Because *lwork* = 0 and *lrwork* = 0, PZGESVD dynamically allocates the work areas used by this subroutine.

### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

      JOBU JOBVT M   N   A IA JA  DESC_A  S   U IU JU  DESC_U  VT  IVT JVT  DESC_VT  WORK  LWORK  RWORK  LRWORK  INFO
      |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
CALL PZGESVD( 'V', 'V', 3, 4, A, 1, 1, DESC_A, S, U, 1, 1, DESC_U, VT, 1, 1, DESC_VT, WORK, 0, RWORK, 0, INFO )
```

	DESC_A	DESC_U	DESC_VT
DTYPE_	1	1	1
CTXT_	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>	<i>icontxt</i> <sup>1</sup>
M_	3	3	3
N_	4	3	4
MB_	2	2	2
NB_	2	2	2
RSRC_	0	0	0
CSRC_	0	0	0
LLD_	See below <sup>2</sup>	See below <sup>2</sup>	See below <sup>2</sup>

### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:

```
LLD_A = MAX(1, NUMROC(M_A, MB_A, MYROW, RSRC_A, NPROW))
LLD_U = MAX(1, NUMROC(M_U, MB_U, MYROW, RSRC_U, NPROW))
LLD_VT = MAX(1, NUMROC(M_VT, MB_VT, MYROW, RSRC_VT, NPROW))
```

In this example:

- LLD\_A = 2 on P<sub>00</sub> and P<sub>01</sub> and LLD\_VT = 1 on P<sub>10</sub> and P<sub>11</sub>
- LLD\_U = 2 on P<sub>00</sub> and P<sub>01</sub> and LLD\_VT = 1 on P<sub>10</sub> and P<sub>11</sub>
- LLD\_VT = 2 on P<sub>00</sub> and P<sub>01</sub> and LLD\_VT = 1 on P<sub>10</sub> and P<sub>11</sub>

Global complex general matrix *A* of size  $3 \times 4$  with block sizes  $2 \times 2$ :

B,D	0	1
0	$\begin{pmatrix} (1.00, 1.00) & (1.00, -1.00) \\ (-1.00, 1.00) & (-1.00, 0.00) \end{pmatrix}$	$\begin{pmatrix} (2.00, 0.00) & (0.00, -2.00) \\ (0.00, -2.00) & (2.00, 2.00) \end{pmatrix}$
1	$\begin{pmatrix} (0.00, -1.00) & (-1.00, -1.00) \end{pmatrix}$	$\begin{pmatrix} (-2.00, 2.00) & (2.00, -2.00) \end{pmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0	1
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$

Local arrays for  $A$ :

p,q	0	1
0	$\begin{pmatrix} (1.00, 1.00) & (1.00, -1.00) \\ (-1.00, 1.00) & (-1.00, 0.00) \end{pmatrix}$	$\begin{pmatrix} (2.00, 0.00) & (0.00, -2.00) \\ (0.00, -2.00) & (2.00, 2.00) \end{pmatrix}$
1	$\begin{pmatrix} (0.00, -1.00) & (-1.00, -1.00) \end{pmatrix}$	$\begin{pmatrix} (-2.00, 2.00) & (2.00, -2.00) \end{pmatrix}$

**Output:**

Global vector  $s$  of length 3:

•  $s = (5.51, 3.31, 2.15)$

Global general matrix  $U$  of size  $3 \times 3$  with block sizes  $2 \times 2$ :

B,D	0	1
0	$\begin{pmatrix} (0.34, 0.00) & (-0.82, 0.00) \\ (-0.60, -0.11) & (-0.06, -0.37) \end{pmatrix}$	$\begin{pmatrix} (-0.45, 0.00) \\ (-0.35, 0.60) \end{pmatrix}$
1	$\begin{pmatrix} (0.08, 0.71) & (0.34, 0.25) \end{pmatrix}$	$\begin{pmatrix} (-0.55, 0.08) \end{pmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0	1
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$

Local arrays for  $U$ :

p,q	0	1
0	$\begin{pmatrix} (0.34, 0.00) & (-0.82, 0.00) \\ (-0.60, -0.11) & (-0.06, -0.37) \end{pmatrix}$	$\begin{pmatrix} (-0.45, 0.00) \\ (-0.35, 0.60) \end{pmatrix}$
1	$\begin{pmatrix} (0.08, 0.71) & (0.34, 0.25) \end{pmatrix}$	$\begin{pmatrix} (-0.55, 0.08) \end{pmatrix}$

Global general matrix  $VT$  of size  $3 \times 4$  with block sizes  $2 \times 2$ :

B,D	0	1
0	$\begin{pmatrix} (0.02, -0.08) & (0.03, 0.03) \\ (-0.42, -0.48) & (-0.41, 0.11) \end{pmatrix}$	$\begin{pmatrix} (0.39, 0.51) & (-0.48, -0.59) \\ (-0.32, 0.39) & (-0.21, 0.33) \end{pmatrix}$
1	$\begin{pmatrix} (0.20, 0.16) & (0.17, 0.78) \end{pmatrix}$	$\begin{pmatrix} (-0.39, -0.11) & (-0.35, -0.03) \end{pmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0	1
0	$P_{00}$	$P_{01}$
1	$P_{10}$	$P_{11}$

Local arrays for  $VT$ :

p,q	0	1
0	( 0.02, -0.08) ( 0.03, 0.03) (-0.42, -0.48) (-0.41, 0.11)	( 0.39, 0.51) (-0.48, -0.59) (-0.32, 0.39) (-0.21, 0.33)
1	( 0.20, 0.16) ( 0.17, 0.78)	(-0.39, -0.11) (-0.35, -0.03)

The value of *info* is 0 on all processes.



---

## Chapter 10. Fourier Transforms

This chapter describes the Fourier Transforms subroutines.

---

### Overview of the Fourier Transforms Subroutines

The Fourier transform subroutines perform mixed-radix transforms in two and three dimensions. See references [1] and [3].

*Table 123. List of Fourier Transform Subroutines*

Descriptive Name	Short- Precision Subroutine	Long- Precision Subroutine	Page
Complex Fourier Transforms in Two Dimensions	PSCFT2	PDCFT2	800
Real-to-Complex Fourier Transforms in Two Dimensions	PSRCFT2	PDRCFT2	807
Complex-to-Real Fourier Transforms in Two Dimensions	PSCRFT2	PDCRFT2	812
Complex Fourier Transforms in Three Dimensions	PSCFT3	PDCFT3	817
Real-to-Complex Fourier Transforms in Three Dimensions	PSRCFT3	PDRCFT3	825
Complex-to-Real Fourier Transforms in Three Dimensions	PSCRFT3	PDCRFT3	831

### Acceptable Lengths for the Transforms

Use the following formula to determine acceptable transform lengths:

$$\bullet \quad n = (2^h) (3^i) (5^j) (7^k) (11^m) \quad \text{for } n \leq 37748736$$

where:

- $h = 1, 2, \dots, 25$
- $i = 0, 1, 2$
- $j, k, m = 0, 1$

Figure 9 on page 798 lists all the acceptable values for transform lengths in the Fourier transform subroutines.

## PSCFT2 and PDCFT2

2	4	6	8	10	12	14	16	18
20	22	24	28	30	32	36	40	42
44	48	56	60	64	66	70	72	80
84	88	90	96	110	112	120	126	128
132	140	144	154	160	168	176	180	192
198	210	220	224	240	252	256	264	280
288	308	320	330	336	352	360	384	396
420	440	448	462	480	504	512	528	560
576	616	630	640	660	672	704	720	768
770	792	840	880	896	924	960	990	1008
1024	1056	1120	1152	1232	1260	1280	1320	1344
1386	1408	1440	1536	1540	1584	1680	1760	1792
1848	1920	1980	2016	2048	2112	2240	2304	2310
2464	2520	2560	2640	2688	2772	2816	2880	3072
3080	3168	3360	3520	3584	3696	3840	3960	4032
4096	4224	4480	4608	4620	4928	5040	5120	5280
5376	5544	5632	5760	6144	6160	6336	6720	6930
7040	7168	7392	7680	7920	8064	8192	8448	8960
9216	9240	9856	10080	10240	10560	10752	11088	11264
11520	12288	12320	12672	13440	13860	14080	14336	14784
15360	15840	16128	16384	16896	17920	18432	18480	19712
20160	20480	21120	21504	22176	22528	23040	24576	24640
25344	26880	27720	28160	28672	29568	30720	31680	32256
32768	33792	35840	36864	36960	39424	40320	40960	42240
43008	44352	45056	46080	49152	49280	50688	53760	55440
56320	57344	59136	61440	63360	64512	65536	67584	71680
73728	73920	78848	80640	81920	84480	86016	88704	90112
92160	98304	98560	101376	107520	110880	112640	114688	118272
122880	126720	129024	131072	135168	143360	147456	147840	157696
161280	163840	168960	172032	177408	180224	184320	196608	197120
202752	215040	221760	225280	229376	236544	245760	253440	258048
262144	270336	286720	294912	295680	315392	322560	327680	337920
344064	354816	360448	368640	393216	394240	405504	430080	443520
450560	458752	473088	491520	506880	516096	524288	540672	573440
589824	591360	630784	645120	655360	675840	688128	709632	720896
737280	786432	788480	811008	860160	887040	901120	917504	946176
983040	1013760	1032192	1048576	1081344	1146880	1179648	1182720	1261568
1290240	1310720	1351680	1376256	1419264	1441792	1474560	1572864	1576960
1622016	1720320	1774080	1802240	1835008	1892352	1966080	2027520	2064384
2097152	2162688	2293760	2359296	2365440	2523136	2580480	2621440	2703360
2752512	2838528	2883584	2949120	3145728	3153920	3244032	3440640	3548160
3604480	3670016	3784704	3932160	4055040	4128768	4194304	4325376	4587520
4718592	4730880	5046272	5160960	5242880	5406720	5505024	5677056	5767168
5898240	6291456	6307840	6488064	6881280	7096320	7208960	7340032	7569408
7864320	8110080	8257536	8388608	8650752	9175040	9437184	9461760	10092544
10321920	10485760	10813440	11010048	11354112	11534336	11796480	12582912	12615680
12976128	13762560	14192640	14417920	14680064	15138816	15728640	16220160	16515072
16777216	17301504	18350080	18874368	18923520	20185088	20643840	20971520	21626880
22020096	22708224	23068672	23592960	25165824	25231360	25952256	27525120	28385280
28835840	29360128	30277632	31457280	32440320	33030144	33554432	34603008	36700160
37748736								

Figure 9. Table of Acceptable Lengths for the Transforms



---

## Fourier Transforms Subroutines

This section contains the Fourier transform subroutine descriptions.

## PSCFT2 and PDCFT2 — Complex Fourier Transforms in Two Dimensions

### Purpose

These subroutines compute the mixed-radix two-dimensional discrete Fourier transform of complex data:

$$y_{k1,k2} = scale \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} x_{j1,j2} W_{n1}^{(Isign)j1k1} W_{n2}^{(Isign)j2k2}$$

for:

- $k1 = 0, 1, \dots, n1-1$
- $k2 = 0, 1, \dots, n2-1$

where:

$$W_{n1} = e^{-2\pi(\sqrt{-1})/n1}$$

$$W_{n2} = e^{-2\pi(\sqrt{-1})/n2}$$

and where:

- $x_{j1,j2}$  are elements of array  $X$ .
- $y_{k1,k2}$  are elements of array  $Y$ .
- $Isign$  is + or – (determined by argument  $isign$ ).
- $scale$  is a scalar value.

For  $scale = 1$  and  $isign$  being positive, you obtain the discrete Fourier transform. For  $scale = 1/((n1)(n2))$  and  $isign$  being negative, you obtain the inverse Fourier transform.

See references [1] and [3].

Table 124. Data Types

$X, Y$	$scale$	Subroutine
Short-precision complex	Short-precision real	PSCFT2
Long-precision complex	Long-precision real	PDCFT2

### Syntax

<b>Fortran</b>	CALL PSCFT2   PDCFT2 ( $x, y, n1, n2, isign, scale, icontxt, ip$ )
<b>C and C++</b>	pscft2   pdcft2 ( $x, y, n1, n2, isign, scale, icontxt, ip$ );

### On Entry

$x$  is the local array  $X$ , containing the two-dimensional data to be transformed that has been block-column distributed over a  $1 \times q$  process grid, where  $q$  is the number of processes. (The value of  $ldx$  is set in the  $IP$  array.)

Scope: **local**

Specified as: an array of (at least) length  $ldx \times \text{LOCq}(n2)$ , containing numbers of the data type indicated in Table 124 on page 800. This array must be aligned on a doubleword boundary.

*y* See On Return.

*n1* is the length of the first dimension of the two-dimensional data in the array to be transformed.

Scope: **global**

Specified as: a fullword integer;  $n1 \leq 37748736$  and must be one of the values listed in Figure 9 on page 798.

*n2* is the length of the second dimension of the two-dimensional data in the array to be transformed.

Scope: **global**

Specified as: a fullword integer;  $n2 \leq 37748736$  and must be one of the values listed in Figure 9 on page 798.

*isign* controls the direction of the transform, determining the sign, *isign*, of the exponent of  $W_m$ , where:

If *isign* = positive value,  $Isign = +$  (transforming time to frequency).

If *isign* = negative value,  $Isign = -$  (transforming frequency to time).

Scope: **global**

Specified as: a fullword integer; where  $isign > 0$  or  $isign < 0$ .

*scale* is the scaling constant *scale*.

Scope: **global**

Specified as: a number of the data type indicated in Table 124 on page 800, where  $scale > 0.0$  or  $scale < 0.0$ .

*icontxt* is the BLACS context parameter.

Scope: **global**

Specified as: the fullword integer that was returned by a prior call to BLACS\_GRIDINIT or BLACS\_GRIDMAP.

*ip* is an array of parameters,  $IP(i)$ , where:

- $IP(1)$  indicates whether the default values for *ip* are used or you set the values for *ip*.

If  $IP(1) = 0$ , then the following default values are used:

- *y* is returned in transposed form; that is, global *y* has dimensions  $n2 \times n1$
- *ldx*, the leading dimension of the array specified for X, equals *n1*
- *ldy*, the leading dimension of the array specified for Y, equals *n2*

The remaining parameters of the array IP are ignored.

If  $IP(1) \neq 0$ , then you set the remaining values of *ip* to indicate whether *y* is stored in normal or transposed form, and indicate values for *ldx* and *ldy*.

- $IP(2)$  indicates whether *y* is to be stored in normal or transposed form.

If  $IP(2) = 0$ , then *y* is to be stored in transposed form on output.

If  $IP(2) = 1$ , then *y* is to be stored in normal form on output.

- $IP(3-19)$  are reserved.

- IP(20) indicates the value of the leading dimension,  $ldx$ , of the array specified for X, where:  
If  $IP(20) = 0$ , then  $ldx = n1$ .  
If  $IP(20) \neq 0$ , then  $ldx$  is this value of IP(20).
- IP(21) indicates the value of the leading dimension,  $ldy$ , of the array specified for Y, where:  
If  $IP(21) = 0$  and  $y$  is to be stored in normal form, then  $ldy = n1$ .  
If  $IP(21) = 0$  and  $y$  is to be stored in transposed form, then  $ldy = n2$ .  
If  $IP(21) \neq 0$ , then  $ldy$  is this value of IP(21).
- IP(22-40) are reserved.

Scope: **global**

Specified as: a one-dimensional array of (at least) length 40, containing fullword integers, where:

- IP(1) is any integer
- IP(2) = 0 or 1
- $IP(20) \geq n1$  or  $IP(20) = 0$
- $IP(21) \geq n1$  (for normal form) or  $IP(21) = 0$
- $IP(21) \geq n2$  (for transposed form) or  $IP(21) = 0$

### On Return

$y$  is the local array Y that is block-column distributed and contains the results of the computation, where:

If  $IP(1) = 0$ , the local array Y is stored in transposed form and has dimensions  $n2 \times LOCq(n1)$ .

If  $IP(1) \neq 0$  and  $IP(2) = 0$ , the local array Y is stored in transposed form and has dimensions  $ldy \times LOCq(n1)$ .

If  $IP(1) \neq 0$  and  $IP(2) = 1$ , the local array Y is stored in normal form and has dimensions  $ldy \times LOCq(n2)$ .

Scope: **local**

Returned as: an  $ldy \times LOCq(n2)$  array (for normal form) or an  $ldy \times LOCq(n1)$  array (for transposed form), containing the numbers of the data type indicated in Table 124 on page 800. This array must be aligned on a doubleword boundary.

### Notes and Coding Rules

1. You may specify the same array for both X and Y. In this case, output overwrites input. If you specify different arrays X and Y, they must have no common elements; otherwise, results are unpredictable.
2. For the output array Y, these subroutines may use any extra space available when  $ldy$  is greater than its minimum value.
3. For more information on LOCq(.) and how sequences are block-column distributed, see "Two-Dimensional Sequence" on page 64.

In general, distributing your data evenly provides the best work load balance among the processes and allows the use of the most efficient collective communication. However, for your specific problem size and number of processes available, experimentation is necessary to achieve optimal performance.

4. An example of the use of this subroutine in a thermal diffusion application program is shown in Appendix B, "Sample Programs." See subroutine `fourier` in "Module Fourier" on page 911.

## Error Conditions

### Computational Errors

None

### Resource Errors

1. Unable to allocate work space

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. `icontxt` is invalid

#### Stage 2:

1. Process grid is not  $1 \times q$
2. The subroutine was called from outside the process grid.

#### Stage 3:

1.  $n1 > 37748736$
2.  $n2 > 37748736$
3. The length of  $n1$  or  $n2$  is not an allowable transform length.
4.  $isign = 0$
5.  $scale = 0.0$
6.  $IP(1) \neq 0$  and  $IP(2) \neq 0$  or 1
7.  $IP(1) \neq 0$  and  $IP(20) \neq 0$  and  $IP(20) < n1$  (that is,  $ldx < n1$ )
8.  $IP(1) \neq 0$  and  $IP(2) = 1$  (for normal mode) and  $IP(21) \neq 0$  and  $IP(21) < n1$  (that is,  $ldy < n1$ )
9.  $IP(1) \neq 0$  and  $IP(2) = 0$  (for transpose mode) and  $IP(21) \neq 0$  and  $IP(21) < n2$  (that is,  $ldy < n2$ )

## Examples

### Example 1

This example shows how to compute a two-dimensional transform. In this example, the `IP` array is set to 0, which means array `Y` is returned in transposed form,  $ldx=n1$ , and  $ldy=n2$ . The data is block-column distributed over a  $1 \times 2$  process grid. The arrays are declared as follows:

```
COMPLEX*16 X(0:7,0:2), Y(0:5,0:3)
INTEGER*4  IP(40)
REAL*8     SCALE
```

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 1
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)
IP(1) = 0
```

```

      X  Y  N1 N2 ISIGN  SCALE  ICONTXT  IP
      |  |  |  |  |      |      |      |
CALL PDCFT2( X , Y , 8 , 6 , -1 , 1.0D0/48.0D0 , ICONTXT , IP)
```

## PSCFT2 and PDCFT2

Global matrix  $X$  of order  $8 \times 6$ :

B,D	0			1		
0	(48.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)
	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)
	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)
	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)
	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)
	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)
	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)

The following is the  $1 \times 2$  process grid:

B,D	0	1
0	$P_{00}$	$P_{01}$

Local arrays for  $X$ :

p,q	0			1		
0	(48.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)
	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)
	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)
	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)
	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)
	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)
	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)

**Output:** Global matrix for  $Y$ :

B,D	0				1			
0	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)
	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)
	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)
	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)
	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)
	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)

The following is the  $1 \times 2$  process grid:

B,D	0	1
0	$P_{00}$	$P_{01}$

Local matrix for  $Y$ :

p,q	0				1			
0	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)
	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)
	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)
	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)
	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)
	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)

## Example 2

This example shows how to compute a two-dimensional transform. This is an example of uneven block-column distribution over a  $1 \times 3$  process grid. In this example, the IP array is set to 0, which means array Y is returned in transposed form,  $ldx=n1$ , and  $ldy=n2$ . The arrays are declared as follows:

```
COMPLEX*16 X(0:7,0:2), Y(0:7,0:2)
INTEGER*4  IP(40)
REAL*8     SCALE
```

### Call Statements and Input:

```
ORDER = 'R'
NPROW = 1
NPCOL = 3
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)
IP(1) = 0
```

```

          X   Y   N1  N2  ISIGN      SCALE      ICONTXT  IP
          |   |   |   |   |         |         |         |
CALL PDCFT2( X , Y , 8 , 8 ,   1 , 1.0D0/16.0D0 , ICONTXT , IP)
```

Global matrix X of order  $8 \times 8$ :

B,D	0			1			2	
0	(0.0,98.0)	(67.0,27.0)	(67.0,82.0)	(84.0,99.0)	(26.0,41.0)	(24.0,15.0)	(27.0,55.0)	(48.0,9.0)
	(13.0,49.0)	(93.0,91.0)	(0.0,12.0)	(52.0,88.0)	(4.0,84.0)	(98.0,57.0)	(43.0,89.0)	(89.0,27.0)
	(75.0,26.0)	(38.0,52.0)	(38.0,1.0)	(9.0,23.0)	(73.0,26.0)	(72.0,80.0)	(76.0,62.0)	(90.0,0.0)
	(45.0,9.0)	(51.0,46.0)	(6.0,68.0)	(65.0,30.0)	(32.0,41.0)	(75.0,3.0)	(47.0,84.0)	(6.0,41.0)
	(53.0,94.0)	(83.0,94.0)	(41.0,86.0)	(41.0,35.0)	(63.0,53.0)	(65.0,53.0)	(23.0,15.0)	(90.0,2.0)
	(21.0,7.0)	(3.0,5.0)	(68.0,62.0)	(70.0,51.0)	(75.0,46.0)	(7.0,49.0)	(27.0,21.0)	(50.0,70.0)
	(4.0,50.0)	(5.0,76.0)	(58.0,73.0)	(91.0,59.0)	(99.0,28.0)	(63.0,95.0)	(35.0,71.0)	(51.0,93.0)
	(67.0,38.0)	(52.0,77.0)	(93.0,72.0)	(76.0,84.0)	(36.0,17.0)	(88.0,74.0)	(16.0,13.0)	(31.0,23.0)

The following is the  $1 \times 3$  process grid:

B,D	0	1	2
-----	-----	-----	-----
0	P <sub>00</sub>	P <sub>01</sub>	P <sub>02</sub>

Local arrays for X:

p,q	0			1			2	
0	(0.0,98.0)	(67.0,27.0)	(67.0,82.0)	(84.0,99.0)	(26.0,41.0)	(24.0,15.0)	(27.0,55.0)	(48.0,9.0)
	(13.0,49.0)	(93.0,91.0)	(0.0,12.0)	(52.0,88.0)	(4.0,84.0)	(98.0,57.0)	(43.0,89.0)	(89.0,27.0)
	(75.0,26.0)	(38.0,52.0)	(38.0,1.0)	(9.0,23.0)	(73.0,26.0)	(72.0,80.0)	(76.0,62.0)	(90.0,0.0)
	(45.0,9.0)	(51.0,46.0)	(6.0,68.0)	(65.0,30.0)	(32.0,41.0)	(75.0,3.0)	(47.0,84.0)	(6.0,41.0)
	(53.0,94.0)	(83.0,94.0)	(41.0,86.0)	(41.0,35.0)	(63.0,53.0)	(65.0,53.0)	(23.0,15.0)	(90.0,2.0)
	(21.0,7.0)	(3.0,5.0)	(68.0,62.0)	(70.0,51.0)	(75.0,46.0)	(7.0,49.0)	(27.0,21.0)	(50.0,70.0)
	(4.0,50.0)	(5.0,76.0)	(58.0,73.0)	(91.0,59.0)	(99.0,28.0)	(63.0,95.0)	(35.0,71.0)	(51.0,93.0)
	(67.0,38.0)	(52.0,77.0)	(93.0,72.0)	(76.0,84.0)	(36.0,17.0)	(88.0,74.0)	(16.0,13.0)	(31.0,23.0)

**Output:** Global matrix for Y:

## PSCFT2 and PDCFT2

B,D	0			1			2	
0	(198.6,200.1)	(-10.6,9.8)	(0.8,7.2)	(5.8,-5.2)	(11.2,9.1)	(-38.3,-18.7)	(-10.2,-1.9)	(14.0,12.6)
	(-0.3,-6.8)	(19.3,-18.7)	(28.7,-3.6)	(-7.2,2.5)	(1.5,14.6)	(-22.0,-20.7)	(29.8,-15.0)	(-10.7,0.8)
	(11.3,-6.2)	(-24.0,-8.1)	(8.6,11.6)	(-29.9,6.5)	(13.7,13.5)	(-16.7,-4.4)	(-26.6,-0.8)	(-3.3,9.5)
	(5.7,17.1)	(3.7,-7.0)	(-2.5,13.9)	(-19.5,-15.9)	(-18.4,20.1)	(11.6,-1.8)	(-0.3,-8.2)	(26.8,30.0)
	(-29.8,-3.4)	(-0.5,7.4)	(-17.1,27.5)	(18.5,32.6)	(9.4,9.6)	(7.6,-8.0)	(-13.1,13.9)	(-26.6,-16.5)
	(-10.2,1.6)	(-5.0,28.8)	(-5.0,25.0)	(5.0,12.1)	(-13.5,9.9)	(2.5,0.6)	(0.0,-5.6)	(-11.8,-8.3)
	(-8.7,-13.6)	(10.0,11.1)	(0.6,9.4)	(12.2,-21.2)	(-9.3,-0.9)	(14.5,-15.6)	(2.4,11.1)	(-22.7,0.2)
	(-27.7,-3.1)	(-21.8,-21.3)	(-22.6,6.0)	(0.2,11.6)	(-1.6,6.6)	(-7.2,-0.4)	(0.5,25.6)	(20.3,23.8)

The following is the  $1 \times 3$  process grid:

B,D	0	1	2
0	P <sub>00</sub>	P <sub>01</sub>	P <sub>02</sub>

Local matrix for Y:

p,q	0			1			2	
0	(198.6,200.1)	(-10.6,9.8)	(0.8,7.2)	(5.8,-5.2)	(11.2,9.1)	(-38.3,-18.7)	(-10.2,-1.9)	(14.0,12.6)
	(-0.3,-6.8)	(19.3,-18.7)	(28.7,-3.6)	(-7.2,2.5)	(1.5,14.6)	(-22.0,-20.7)	(29.8,-15.0)	(-10.7,0.8)
	(11.3,-6.2)	(-24.0,-8.1)	(8.6,11.6)	(-29.9,6.5)	(13.7,13.5)	(-16.7,-4.4)	(-26.6,-0.8)	(-3.3,9.5)
	(5.7,17.1)	(3.7,-7.0)	(-2.5,13.9)	(-19.5,-15.9)	(-18.4,20.1)	(11.6,-1.8)	(-0.3,-8.2)	(26.8,30.0)
	(-29.8,-3.4)	(-0.5,7.4)	(-17.1,27.5)	(18.5,32.6)	(9.4,9.6)	(7.6,-8.0)	(-13.1,13.9)	(-26.6,-16.5)
	(-10.2,1.6)	(-5.0,28.8)	(-5.0,25.0)	(5.0,12.1)	(-13.5,9.9)	(2.5,0.6)	(0.0,-5.6)	(-11.8,-8.3)
	(-8.7,-13.6)	(10.0,11.1)	(0.6,9.4)	(12.2,-21.2)	(-9.3,-0.9)	(14.5,-15.6)	(2.4,11.1)	(-22.7,0.2)
	(-27.7,-3.1)	(-21.8,-21.3)	(-22.6,6.0)	(0.2,11.6)	(-1.6,6.6)	(-7.2,-0.4)	(0.5,25.6)	(20.3,23.8)



## PSRCFT2 and PDRCFT2 — Real-to-Complex Fourier Transforms in Two Dimensions

### Purpose

These subroutines compute the mixed-radix two-dimensional complex conjugate even discrete Fourier transform of real data:

$$y_{k1,k2} = scale \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} x_{j1,j2} W_{n1}^{(Isign)j1k1} W_{n2}^{(Isign)j2k2}$$

for:

- $k1 = 0, 1, \dots, n1-1$
- $k2 = 0, 1, \dots, n2-1$

where:

$$W_{n1} = e^{-2\pi(\sqrt{-1})/n1}$$

$$W_{n2} = e^{-2\pi(\sqrt{-1})/n2}$$

and where:

- $x_{j1,j2}$  are elements of array  $X$ .
- $y_{k1,k2}$  are elements of array  $Y$ .
- $Isign$  is + or – (determined by argument  $isign$ ).
- $scale$  is a scalar value.

For  $scale = 1$  and  $isign$  being positive, you obtain the discrete Fourier transform. For  $scale = 1/((n1)(n2))$  and  $isign$  being negative, you obtain the inverse Fourier transform.

See references [1] and [3].

Table 125. Data Types

$X, scale$	$Y$	Subroutine
Short-precision real	Short-precision complex	PSRCFT2
Long-precision real	Long-precision complex	PDRCFT2

### Syntax

<b>Fortran</b>	CALL PSRCFT2   PDRCFT2 ( $x, y, n1, n2, isign, scale, icontxt, ip$ )
<b>C and C++</b>	psrcft2   pdrctf2 ( $x, y, n1, n2, isign, scale, icontxt, ip$ );

### On Entry

$x$  is the local array  $X$ , containing the two-dimensional data to be transformed that has been block-column distributed over a  $1 \times q$  process grid, where  $q$  is the number of processes. (The value of  $ldx$  is set in the  $IP$  array.)

Scope: **local**

Specified as: an array of (at least) length  $ldx \times LOCq(n2)$ , containing numbers of the data type indicated in Table 125 on page 807. This array must be aligned on a doubleword boundary.

*y* See On Return.

*n1* is the length of the first dimension of the two-dimensional data in the array to be transformed.

Scope: **global**

Specified as: a fullword integer;  $n1 \leq 37748736$  and must be one of the values listed in Figure 9 on page 798.

*n2* is the length of the second dimension of the two-dimensional data in the array to be transformed.

Scope: **global**

Specified as: a fullword integer;  $n2 \leq 37748736$  and must be one of the values listed in Figure 9 on page 798.

*isign* controls the direction of the transform, determining the sign, *isign*, of the exponent of  $W_m$ , where:

If *isign* = positive value,  $Isign = +$  (transforming time to frequency).

If *isign* = negative value,  $Isign = -$  (transforming frequency to time).

Scope: **global**

Specified as: a fullword integer; where  $isign > 0$  or  $isign < 0$ .

*scale* is the scaling constant *scale*.

Scope: **global**

Specified as: a number of the data type indicated in Table 125 on page 807, where  $scale > 0.0$  or  $scale < 0.0$ .

*icontxt* is the BLACS context parameter.

Scope: **global**

Specified as: the fullword integer that was returned by a prior call to BLACS\_GRIDINIT or BLACS\_GRIDMAP.

*ip* is an array of parameters,  $IP(i)$ , where:

- $IP(1)$  indicates whether the default values for *ip* are used or you set the values for *ip*.

If  $IP(1) = 0$ , then the following default values are used:

- *ldx*, the leading dimension of the array specified for X, equals *n1*
- *ldy*, the leading dimension of the array specified for Y, equals *n2*

The remaining parameters of the array IP are ignored.

If  $IP(1) \neq 0$ , then you set the remaining values of *ip* to indicate values for *ldx* and *ldy*.

- $IP(2-19)$  are reserved.
- $IP(20)$  indicates the value of the leading dimension, *ldx*, of the array specified for X, where:
  - If  $IP(20) = 0$ , then  $ldx = n1$ .
  - If  $IP(20) \neq 0$ , then *ldx* is this value of  $IP(20)$ .
- $IP(21)$  indicates the value of the leading dimension, *ldy*, of the array specified for Y, where:

If  $IP(21) = 0$ , then  $ldy = n2$ .

If  $IP(21) \neq 0$ , then  $ldy$  is this value of  $IP(21)$ .

- $IP(22-40)$  are reserved.

Scope: **global**

Specified as: a one-dimensional array of (at least) length 40, containing fullword integers, where:

$IP(1)$  is any integer

$IP(20) \geq n1$  or  $IP(20) = 0$

$IP(21) \geq n2$  or  $IP(21) = 0$

### On Return

$y$  is the local array  $Y$ , stored in FFT-packed storage mode, containing the results of the computation that are block-column distributed, where:

If  $IP(1) = 0$ , the local array  $Y$  has dimensions  $n2 \times LOCq(n1/2)$ .

If  $IP(1) \neq 0$  and  $IP(21) = 0$ , the local array  $Y$  has dimensions  $n2 \times LOCq(n1/2)$ .

If  $IP(1) \neq 0$  and  $IP(21) \neq 0$ , the local array  $Y$  has dimensions  $ldy \times LOCq(n1/2)$ .

Scope: **local**

Returned as: an  $ldy \times LOCq(n1/2)$  array, containing the numbers of the data type indicated in Table 125 on page 807. This array must be aligned on a doubleword boundary.

## Notes and Coding Rules

1. These subroutines always return  $Y$  in transposed form.
2. For the output array  $Y$ , these subroutines may use any extra space available when  $ldy$  is greater than its minimum value.
3. You may specify the same array for  $X$  and  $Y$ . In this case, output overwrites input. If you specify different arrays  $X$  and  $Y$ , they must have no common elements; otherwise, results are unpredictable.
4. For more information on  $LOCq(\_)$ , and how sequences are block-column distributed and stored in FFT-packed storage mode, see "Two-Dimensional Sequence" on page 64.

In general, distributing your data evenly provides the best work load balance among the processes and allows the use of the most efficient collective communication. However, for your specific problem size and number of processes available, experimentation is necessary to achieve optimal performance.

## Error Conditions

### Computational Errors

None

### Resource Errors

1. Unable to allocate work space

## Input-Argument and Miscellaneous Errors

### Stage 1:

1. *icontxt* is invalid

### Stage 2:

1. Process grid is not  $1 \times q$
2. The subroutine was called from outside the process grid.

### Stage 3:

1.  $n1 > 37748736$
2.  $n2 > 37748736$
3. The length of  $n1$  or  $n2$  is not an allowable transform length.
4.  $isign = 0$
5.  $scale = 0.0$
6.  $IP(1) \neq 0$  and  $IP(20) \neq 0$  and  $IP(20) < n1$  (that is,  $ldx < n1$ )
7.  $IP(1) \neq 0$  and  $IP(21) \neq 0$  and  $IP(21) < n2$  (that is,  $ldy < n2$ )

## Examples

### Example

This example shows how to compute a two-dimensional transform. The data is block-column distributed over a  $1 \times 2$  process grid. The arrays are declared as follows:

```
REAL*8 X(0:11,0:1)
COMPLEX*16 Y(0:6,0:1)
INTEGER*4 IP(40)
REAL*8 SCALE
```

### Call Statements and Input:

```
ORDER = 'R'
NPROW = 1
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)
IP(1) = 1
IP(20) = 12 (that is,  $ldx = 12$ )
IP(21) = 7 (that is,  $ldy = 7$ )
```

```

           X   Y   N1  N2  ISIGN  SCALE  ICONTXT  IP
           |   |   |   |   |       |       |   |
CALL PDRCFT2( X , Y , 8 , 4 ,   1 , 1.0D0 , ICONTXT , IP)
```

Global matrix  $X$  of order  $8 \times 4$ :

```
B,D      0      1
```

$$0 \begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

The following is the  $1 \times 2$  process grid:

B,D	0	1
0	$P_{00}$	$P_{01}$

Local arrays for  $X$ :

p,q	0	1
0	1.0 0.0	0.0 0.0
	0.0 0.0	0.0 0.0
	0.0 0.0	0.0 0.0
	0.0 0.0	0.0 0.0
	0.0 0.0	0.0 0.0
	0.0 0.0	0.0 0.0
	0.0 0.0	0.0 0.0
	0.0 0.0	0.0 0.0
	0.0 0.0	0.0 0.0
	0.0 0.0	0.0 0.0
	.	.
	.	.
	.	.
	.	.

**Output:** The following global matrix  $Y$  is returned in transposed form and stored in FFT-packed storage mode:

B,D	0	1
0	$\begin{bmatrix} (1.0,1.0) & (1.0,0.0) \\ (1.0,0.0) & (1.0,0.0) \\ (1.0,1.0) & (1.0,0.0) \\ (1.0,0.0) & (1.0,0.0) \\ \vdots & \vdots \\ \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} (1.0,0.0) & (1.0,0.0) \\ (1.0,0.0) & (1.0,0.0) \\ (1.0,0.0) & (1.0,0.0) \\ (1.0,0.0) & (1.0,0.0) \\ \vdots & \vdots \\ \vdots & \vdots \end{bmatrix}$

The following is the  $1 \times 2$  process grid:

B,D	0	1
0	$P_{00}$	$P_{01}$

The following local arrays for  $Y$  are returned in transposed form and stored in FFT-packed storage mode:

p,q	0	1
0	(1.0,1.0)	(1.0,0.0)
	(1.0,0.0)	(1.0,0.0)
	(1.0,1.0)	(1.0,0.0)
	(1.0,0.0)	(1.0,0.0)
	(1.0,1.0)	(1.0,0.0)
	(1.0,0.0)	(1.0,0.0)
	(1.0,1.0)	(1.0,0.0)
	(1.0,0.0)	(1.0,0.0)
	(1.0,1.0)	(1.0,0.0)
	(1.0,0.0)	(1.0,0.0)
	.	.
	.	.
	.	.
	.	.

## PSCRFT2 and PDCRFT2 — Complex-to-Real Fourier Transforms in Two Dimensions

### Purpose

These subroutines compute the mixed-radix two-dimensional real discrete Fourier transform of complex conjugate even data:

$$y_{k1,k2} = scale \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} x_{j1,j2} W_{n1}^{(Isign)j1k1} W_{n2}^{(Isign)j2k2}$$

for:

- $k1 = 0, 1, \dots, n1-1$
- $k2 = 0, 1, \dots, n2-1$

where:

$$W_{n1} = e^{-2\pi(\sqrt{-1})/n1}$$

$$W_{n2} = e^{-2\pi(\sqrt{-1})/n2}$$

and where:

- $x_{j1,j2}$  are elements of array  $X$ .
- $y_{k1,k2}$  are elements of array  $Y$ .
- $Isign$  is + or – (determined by argument  $isign$ ).
- $scale$  is a scalar value.

For  $scale = 1$  and  $isign$  being positive, you obtain the discrete Fourier transform. For  $scale = 1/((n1)(n2))$  and  $isign$  being negative, you obtain the inverse Fourier transform.

See references [1] and [3].

Table 126. Data Types

$X$	$Y, scale$	Subroutine
Short-precision complex	Short-precision real	PSCRFT2
Long-precision complex	Long-precision real	PDCRFT2

### Syntax

<b>Fortran</b>	CALL PSCRFT2   PDCRFT2 ( $x, y, n1, n2, isign, scale, icontxt, ip$ )
<b>C and C++</b>	pscrft2   pdcrt2 ( $x, y, n1, n2, isign, scale, icontxt, ip$ );

### On Entry

$x$  is the local array  $X$ , containing the two-dimensional data to be transformed that has been block-column distributed over a  $1 \times q$  process grid, where  $q$  is the number of processes. (The value of  $ldx$  is set in the  $IP$  array.) Array  $X$  is stored in FFT-packed storage mode.

Scope: **local**

Specified as: an array of (at least) length  $ldx \times \text{LOCq}(n1/2)$ , containing numbers of the data type indicated in Table 126 on page 812. This array must be aligned on a doubleword boundary.

*y* See On Return.

*n1* is the length of the second dimension of the two-dimensional data of the array to be transformed.

Scope: **global**

Specified as: a fullword integer;  $n1 \leq 37748736$  and must be one of the values listed in Figure 9 on page 798.

*n2* is the length of the first dimension of two-dimensional data of the array to be transformed.

Scope: **global**

Specified as: a fullword integer;  $n2 \leq 37748736$  and must be one of the values listed in Figure 9 on page 798.

*isign* controls the direction of the transform, determining the sign, *isign*, of the exponent of  $W_n$ , where:

If *isign* = positive value,  $Isign = +$  (transforming time to frequency).

If *isign* = negative value,  $Isign = -$  (transforming frequency to time).

Scope: **global**

Specified as: a fullword integer; where  $isign > 0$  or  $isign < 0$ .

*scale* is the scaling constant *scale*.

Scope: **global**

Specified as: a number of the data type indicated in Table 126 on page 812, where  $scale > 0.0$  or  $scale < 0.0$ .

*icontxt* is the BLACS context parameter.

Scope: **global**

Specified as: the fullword integer that was returned by a prior call to BLACS\_GRIDINIT or BLACS\_GRIDMAP.

*ip* is an array of parameters,  $IP(i)$ , where:

- $IP(1)$  indicates whether the default values for *ip* are used or you set the values for *ip*.

If  $IP(1) = 0$ , then the following default values are used:

- *ldx*, the leading dimension of the array specified for *X*, equals *n2*
- *ldy*, the leading dimension of the array specified for *Y*, equals *n1*

The remaining parameters of the array *IP* are ignored.

If  $IP(1) \neq 0$ , then you set the remaining values of *ip* to indicate values for *ldx* and *ldy*.

- $IP(2-19)$  are reserved.
- $IP(20)$  indicates the value of the leading dimension, *ldx*, of the array specified for *X*, where:
  - If  $IP(20) = 0$ , then  $ldx = n2$ .
  - If  $IP(20) \neq 0$ , then *ldx* is this value of  $IP(20)$ .

- IP(21) indicates the value of the leading dimension, *ldy*, of the array specified for Y, where:  
If  $IP(21) = 0$  then  $ldy = n1$ .  
If  $IP(21) \neq 0$ , then *ldy* is this value of IP(21).
- IP(22-40) are reserved.

Scope: **global**

Specified as: a one-dimensional array of (at least) length 40, containing fullword integers, where:

IP(1) is any integer

$IP(20) \geq n2$  or  $IP(20) = 0$

$IP(21) \geq n1$  or  $IP(21) = 0$

### On Return

*y* is the local array Y that is block-column distributed and contains the results of the computation, where:

If  $IP(1) = 0$ , the local array Y is stored in normal form and has dimensions  $n1 \times LOCq(n2)$ .

If  $IP(1) \neq 0$  and  $IP(21) = 0$ , the local array Y is stored in normal form and has dimensions  $n1 \times LOCq(n2)$ .

If  $IP(1) \neq 0$  and  $IP(21) \neq 0$ , the local array Y is stored in normal form and has dimensions  $ldy \times LOCq(n2)$ .

Scope: **local**

Returned as: an  $ldy \times LOCq(n2)$  array, containing the numbers of the data type indicated in Table 126 on page 812. This array must be aligned on a doubleword boundary.

## Notes and Coding Rules

1. These subroutines always return Y in normal form.
2. For the output array Y, these subroutines may use any extra space available when *ldy* is greater than its minimum value.
3. For more information on LOCq(\_), and how sequences are block-column distributed and stored in FFT-packed storage mode, see "Two-Dimensional Sequence" on page 64.

In general, distributing your data evenly provides the best work load balance among the processes and allows the use of the most efficient collective communication. However, for your specific problem size and number of processes available, experimentation is necessary to achieve optimal performance.

4. You may specify the same array for X and Y. In this case, output overwrites input. If you specify different arrays X and Y, they must have no common elements; otherwise, results are unpredictable.

## Error Conditions

### Computational Errors

None



## Resource Errors

1. Unable to allocate work space.

## Input-Argument and Miscellaneous Errors

### Stage 1:

1. *icontxt* is invalid

### Stage 2:

1. Process grid is not  $1 \times q$
2. The subroutine was called from outside the process grid.

### Stage 3:

1.  $n1 > 37748736$
2.  $n2 > 37748736$
3. The length of  $n1$  or  $n2$  is not an allowable transform length.
4.  $isign = 0$
5.  $scale = 0.0$
6.  $IP(1) \neq 0$  and  $IP(20) \neq 0$  and  $IP(20) < n2$  (that is,  $ldx < n2$ )
7.  $IP(1) \neq 0$  and  $IP(21) \neq 0$  and  $IP(21) < n1$  (that is,  $ldy < n1$ )

## Examples

### Example

This example shows how to compute a two-dimensional transform. The data is block-column distributed over a  $1 \times 2$  process grid. The arrays are declared as follows:

```
COMPLEX*16 X(0:6,0:1)
REAL*8 Y(0:11,0:1)
INTEGER*4 IP(40)
REAL*8 SCALE
```

### Call Statements and Input:

```
ORDER = 'R'
NPROW = 1
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)
IP(1) = 1
IP(20) = 7 (that is,  $ldx = 7$ )
IP(21) = 12 (that is,  $ldy = 12$ )
```

```

           X   Y   N1 N2 ISIGN   SCALE   ICONTXT   IP
           |   |   |  |  |      |       |         |
CALL PDCRFT2( X , Y , 8 , 4 , -1 , 1.00D0/32.00D0 , ICONTXT , IP)
```

The following global matrix  $X$  is stored in FFT-packed storage mode:

```

B,D          0          1
0  [ (1.0,1.0) (1.0,0.0) | (1.0,0.0) (1.0,0.0)
    (1.0,0.0) (1.0,0.0) | (1.0,0.0) (1.0,0.0)
    (1.0,1.0) (1.0,0.0) | (1.0,0.0) (1.0,0.0)
    (1.0,0.0) (1.0,0.0) | (1.0,0.0) (1.0,0.0)
      .         .         | .         .
      .         .         | .         .
      .         .         | .         .
      .         .         | .         . ]
```

The following is the  $1 \times 2$  process grid:

## PSCRFT2 and PDCRFT2

B,D	0	1
0	$P_{00}$	$P_{01}$

The following local arrays for  $X$  are stored in FFT-packed storage mode:

p,q	0	1
0	$\begin{pmatrix} (1.0,1.0) & (1.0,0.0) \\ (1.0,0.0) & (1.0,0.0) \\ (1.0,1.0) & (1.0,0.0) \\ (1.0,0.0) & (1.0,0.0) \end{pmatrix}$	$\begin{pmatrix} (1.0,0.0) & (1.0,0.0) \\ (1.0,0.0) & (1.0,0.0) \\ (1.0,0.0) & (1.0,0.0) \\ (1.0,0.0) & (1.0,0.0) \end{pmatrix}$
	$\begin{pmatrix} \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \end{pmatrix}$	$\begin{pmatrix} \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \end{pmatrix}$

**Output:** Global matrix  $Y$ :

B,D	0	1
0	$\begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}$	

The following is the  $1 \times 2$  process grid:

B,D	0	1
0	$P_{00}$	$P_{01}$

Local arrays for  $Y$ :

p,q	0	1
0	$\begin{pmatrix} 1.0 & 0.0 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \end{pmatrix}$	$\begin{pmatrix} 0.0 & 0.0 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \end{pmatrix}$

## PSCFT3 and PDCFT3 — Complex Fourier Transforms in Three Dimensions

### Purpose

These subroutines compute the mixed-radix three-dimensional discrete Fourier transform of complex data:

$$y_{k1,k2,k3} = scale \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \sum_{j3=0}^{n3-1} x_{j1,j2,j3} W_{n1}^{(Isign)j1k1} W_{n2}^{(Isign)j2k2} W_{n3}^{(Isign)j3k3}$$

for:

- $k1 = 0, 1, \dots, n1-1$
- $k2 = 0, 1, \dots, n2-1$
- $k3 = 0, 1, \dots, n3-1$

where:

$$W_{n1} = e^{-2\pi(\sqrt{-1})/n1}$$

$$W_{n2} = e^{-2\pi(\sqrt{-1})/n2}$$

$$W_{n3} = e^{-2\pi(\sqrt{-1})/n3}$$

and where:

- $x_{j1,j2,j3}$  are elements of array  $X$ .
- $y_{k1,k2,k3}$  are elements of array  $Y$ .
- $Isign$  is + or – (determined by argument  $isign$ ).
- $scale$  is a scalar value.

For  $scale = 1$  and  $isign$  being positive, you obtain the discrete Fourier transform. For  $scale = 1/((n1)(n2)(n3))$  and  $isign$  being negative, you obtain the inverse Fourier transform.

See references [1] and [3].

Table 127. Data Types

$X, Y$	$scale$	Subroutine
Short-precision complex	Short-precision real	PSCFT3
Long-precision complex	Long-precision real	PDCFT3

### Syntax

<b>Fortran</b>	CALL PSCFT3   PDCFT3 ( $x, y, n1, n2, n3, isign, scale, icontxt, ip$ )
<b>C and C++</b>	pscft3   pdcft3 ( $x, y, n1, n2, n3, isign, scale, icontxt, ip$ );

### On Entry

$x$  is the local array  $X$ , containing the three-dimensional data to be

transformed that has been block-plane distributed over a  $1 \times q$  process grid, where  $q$  is the number of processes. (The values of  $ldx1$  and  $ldx2$  are set in the IP array.)

Scope: **local**

Specified as: an array of (at least) length  $ldx1 \times ldx2 \times \text{LOCq}(n3)$ , containing numbers of the data type indicated in Table 127 on page 817. This array must be aligned on a doubleword boundary.

$y$  See On Return.

$n1$  is the length of the first dimension of the three-dimensional data of the array to be transformed.

Scope: **global**

Specified as: a fullword integer;  $n1 \leq 37748736$  and must be one of the values listed in Figure 9 on page 798.

$n2$  is the length of the second dimension of the three-dimensional data of the array to be transformed.

Scope: **global**

Specified as: a fullword integer;  $n2 \leq 37748736$  and must be one of the values listed in Figure 9 on page 798.

$n3$  is the length of the third dimension of the three-dimensional data of the array to be transformed.

Scope: **global**

Specified as: a fullword integer;  $n3 \leq 37748736$  and must be one of the values listed in Figure 9 on page 798.

$isign$  controls the direction of the transform, determining the sign,  $isign$ , of the exponent of  $W_n$ , where:

If  $isign =$  positive value,  $Isign = +$  (transforming time to frequency).

If  $isign =$  negative value,  $Isign = -$  (transforming frequency to time).

Scope: **global**

Specified as: a fullword integer; where  $isign > 0$  or  $isign < 0$ .

$scale$  is the scaling constant  $scale$ .

Scope: **global**

Specified as: a number of the data type indicated in Table 127 on page 817, where  $scale > 0.0$  or  $scale < 0.0$ .

$icontxt$  is the BLACS context parameter.

Scope: **global**

Specified as: the fullword integer that was returned by a prior call to BLACS\_GRIDINIT or BLACS\_GRIDMAP.

$ip$  is an array of parameters,  $IP(i)$ , where:

- $IP(1)$  indicates whether the default values for  $ip$  are used or you set the values for  $ip$ .

If  $IP(1) = 0$ , then the following default values are used:

- $y$  is returned in transposed form; that is global  $y$  has dimensions  $n3 \times n2 \times n1$ .

- $ldx1$  and  $ldx2$ , the leading dimensions of the array specified for  $X$ , equal  $n1$  and  $n2$ , respectively.
- $ldy1$  and  $ldy2$ , the leading dimensions of the array specified for  $Y$ , equal  $n3$  and  $n2$ , respectively.

The remaining parameters of array  $IP$  are ignored.

If  $IP(1) \neq 0$ , then you set the remaining values of  $ip$  to indicate whether  $y$  is stored in normal or transposed form, and indicate values for the leading dimensions.

- $IP(2)$  indicates whether  $y$  is to be stored in normal or transposed form.  
If  $IP(2)=0$ , then  $y$  is to be stored in transposed form on output.  
If  $IP(2)=1$ , then  $y$  is to be stored in normal form on output.
- $IP(3-19)$  are reserved.
- $IP(20)$  indicates the values of the leading dimension,  $ldx1$ , for the array specified for  $X$ , where:  
If  $IP(20) = 0$ , then  $ldx1 = n1$ .  
If  $IP(20) \neq 0$ , then  $ldx1$  is this value of  $IP(20)$ .
- $IP(21)$  indicates the values of the leading dimension,  $ldx2$ , for the array specified for  $X$ , where:  
If  $IP(21) = 0$ , then  $ldx2 = n2$ .  
If  $IP(21) \neq 0$ , then  $ldx2$  is this value of  $IP(21)$ .
- $IP(22)$  indicates the values of the leading dimension,  $ldy1$ , for the array specified for  $Y$ , where:  
If  $IP(22) = 0$  and  $IP(2) = 1$ , then  $ldy1 = n1$ .  
If  $IP(22) = 0$  and  $IP(2) = 0$ , then  $ldy1 = n3$ .  
If  $IP(22) \neq 0$ , then  $ldy1$  is this value of  $IP(22)$ .
- $IP(23)$  indicates the values of the leading dimension,  $ldy2$ , for the array specified for  $Y$ , where:  
If  $IP(23) = 0$ , then  $ldy2 = n2$ .  
If  $IP(23) \neq 0$ , then  $ldy2$  is this value of  $IP(23)$ .
- $IP(24-40)$  are reserved.

Scope: **global**

Specified as: a one-dimensional array of (at least) length 40, containing fullword integers, where:

- $IP(1)$  is any integer
- $IP(2) = 0$  or  $1$
- $IP(20) \geq n1$  or  $IP(20)=0$
- $IP(21) \geq n2$  or  $IP(21)=0$
- $IP(22) \geq n1$  (for normal form) or  $IP(22)=0$
- $IP(22) \geq n3$  (for transposed form) or  $IP(22) = 0$
- $IP(23) \geq n2$  or  $IP(23)=0$

## On Return

$y$  is the local array  $Y$  that is block-plane distributed and contains the results of the computation, where:

If  $IP(1) = 0$ , the local array  $Y$  is stored in transposed form and has dimensions  $n3 \times n2 \times LOCq(n1)$ .

If  $IP(1) \neq 0$  and  $IP(2)=0$ , then the local array  $Y$  is stored in transposed form and has dimensions  $ldy1 \times ldy2 \times LOCq(n1)$ .

If  $IP(1) \neq 0$  and  $IP(2)=1$ , then the local array  $Y$  is stored in normal form and has dimensions  $ldy1 \times ldy2 \times LOCq(n3)$ .

Scope: **local**

Returned as: an  $ldy1 \times ldy2 \times LOCq(n3)$  array (for normal form) or an  $ldy1 \times ldy2 \times LOCq(n1)$  array (for transposed form), containing the numbers of the data type indicated in Table 127 on page 817. This array must be aligned on a doubleword boundary.

## Notes and Coding Rules

1. For the output array  $Y$ , these subroutines may use any extra space available when  $ldy1$  and  $ldy2$  are greater than their minimum value.
2. You may specify the same array for  $X$  and  $Y$ . In this case, output overwrites input. If you specify different arrays  $X$  and  $Y$ , they must have no common elements; otherwise, results are unpredictable.
3. For more information on  $LOCq(\_)$  and how sequences are block-plane distributed, see “Three-Dimensional Sequences” on page 68.

In general, distributing your data evenly provides the best work load balance among the processes and allows the use of the most efficient collective communication. However, for your specific problem size and number of processes available, experimentation is necessary to achieve optimal performance.

## Error Conditions

### Computational Errors

None

### Resource Errors

1. Unable to allocate work space.

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. *icontxt* is invalid

#### Stage 2:

1. Process grid is not  $1 \times q$
2. The subroutine was called from outside the process grid.

#### Stage 3:

1.  $n1 > 37748736$
2.  $n2 > 37748736$
3.  $n3 > 37748736$
4. The length of  $n1$ ,  $n2$ , or  $n3$  is not an allowable transform length.
5.  $isign = 0$
6.  $scale = 0.0$
7.  $IP(1) \neq 0$  and  $IP(2) \neq 0$  or 1
8.  $IP(1) \neq 0$  and  $IP(20) \neq 0$  and  $IP(20) < n1$  (that is,  $ldx1 < n1$ )
9.  $IP(1) \neq 0$  and  $IP(21) \neq 0$  and  $IP(21) < n2$  (that is,  $ldx2 < n2$ )
10.  $IP(1) \neq 0$  and  $IP(2)=0$  (for transpose mode) and  $IP(22) \neq 0$  and  $IP(22) < n3$  (that is,  $ldy1 < n3$ )
11.  $IP(1) \neq 0$  and  $IP(2)=1$  (for normal mode) and  $IP(22) \neq 0$  and  $IP(22) < n1$  (that is,  $ldy1 < n1$ )
12.  $IP(1) \neq 0$  and  $IP(23) \neq 0$  and  $IP(23) < n2$  (that is,  $ldy2 < n2$ )

## Examples

### Example 1

This example shows how to compute a three-dimensional transform. The data is block-plane distributed over a  $1 \times 2$  process grid. The arrays are declared as follows:

```
COMPLEX*16 X(0:3,0:3,0)
COMPLEX*16 Y(0:3,0:3,0)
INTEGER*4 IP(40)
REAL*8 SCALE
```

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 1
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)
IP(1) = 1
IP(2) = 1
IP(20) = 4
IP(21) = 4
IP(22) = 4
IP(23) = 4
```

```

           X   Y   N1  N2  N3  ISIGN  SCALE  ICONTXT  IP
           |   |   |   |   |   |      |      |
CALL PDCFT3( X , Y , 4 , 4 , 2 , 1 , 1.0D0 , ICONTXT , IP)
```

Global matrix  $X$ :

#### Plane 0:

B,D 0

$$\begin{bmatrix} (1.0,0.0) & (0.0,0.0) & (0.0,0.0) & (0.0,0.0) \\ (0.0,0.0) & (0.0,0.0) & (0.0,0.0) & (0.0,0.0) \\ (0.0,0.0) & (0.0,0.0) & (0.0,0.0) & (0.0,0.0) \\ (0.0,0.0) & (0.0,0.0) & (0.0,0.0) & (0.0,0.0) \end{bmatrix}$$

#### Plane 1:

B,D 1

$$\begin{bmatrix} (0.0,0.0) & (0.0,0.0) & (0.0,0.0) & (0.0,0.0) \\ (0.0,0.0) & (0.0,0.0) & (0.0,0.0) & (0.0,0.0) \\ (0.0,0.0) & (0.0,0.0) & (0.0,0.0) & (0.0,0.0) \\ (0.0,0.0) & (0.0,0.0) & (0.0,0.0) & (0.0,0.0) \end{bmatrix}$$

The following is the  $1 \times 2$  process grid:

B,D	0	1
-----	-----	-----
0	$P_{00}$	$P_{01}$

Local arrays for  $X$ :

## PSCFT3 and PDCFT3

p,q	0				1			
0	(1.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)
	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)
	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)
	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)

**Output:** Global matrix  $Y$ :

**Plane 0:**

B,D	0			
0	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)
	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)
	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)
	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)

**Plane 1:**

B,D	1			
0	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)
	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)
	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)
	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)

The following is the  $1 \times 2$  process grid:

B,D	0	1
0	$P_{00}$	$P_{01}$

Local arrays for  $Y$ :

p,q	0				1			
0	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)
	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)
	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)
	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)

### Example 2

This example shows how to compute a three-dimensional transform. In this example, the IP array is set to 0, which means array  $Y$  is returned in transposed form,  $ldx=n1$ , and  $ldy=n3$ . This is an example of uneven block-plane distribution over a  $1 \times 3$  process grid. The arrays are declared as follows:

```
COMPLEX*16 X(0:3,0:1,0:1), Y(0:5,0:1,0:1)
INTEGER*4  IP(40)
REAL*8     SCALE
```

### Call Statements and Input:

```
ORDER = 'R'
NPROW = 1
NPCOL = 3
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)
IP(1) = 0
```



```

          X   Y   N1  N2  N3  ISIGN   SCALE   ICONXT   IP
          |   |   |   |   |   |       |       |   |
CALL PDCFT3( X , Y , 4 , 2 , 6 , 1 , 1.0D0/8.0D0, ICONXT , IP)

```

Global matrix  $X$ :

	Plane 0:	Plane 1:
B,D	0	
0	$\begin{bmatrix} (4.9,3.9) & (5.9,3.1) & (2.2,5.1) & (5.9,1.5) \\ (6.8,4.6) & (6.7,9.8) & (2.1,0.5) & (3.5,2.9) \\ (4.9,1.9) & (1.6,4.9) & (7.0,6.8) & (6.4,1.1) \\ (2.9,7.6) & (7.5,5.5) & (0.6,4.6) & (9.0,7.6) \end{bmatrix}$	
	Plane 2:	Plane 3:
B,D	1	
0	$\begin{bmatrix} (8.3,2.3) & (7.3,7.3) & (4.6,5.9) & (3.3,3.0) \\ (4.5,8.9) & (0.2,0.9) & (3.6,5.0) & (3.4,0.4) \\ (1.8,6.5) & (2.5,1.7) & (8.5,9.3) & (0.2,1.7) \\ (6.4,5.2) & (7.8,5.3) & (8.7,9.1) & (9.1,5.5) \end{bmatrix}$	
	Plane 4:	Plane 5:
B,D	2	
0	$\begin{bmatrix} (1.0,1.0) & (2.2,8.2) & (5.4,3.6) & (4.8,4.0) \\ (8.8,1.0) & (8.9,5.8) & (2.0,2.6) & (5.2,9.5) \\ (3.3,0.0) & (8.9,7.5) & (3.1,9.7) & (5.4,7.5) \\ (8.7,8.3) & (6.3,5.4) & (1.5,9.6) & (4.4,4.4) \end{bmatrix}$	

The following is the  $1 \times 3$  process grid:

B,D	0	1	2
0	$P_{00}$	$P_{01}$	$P_{02}$

Local arrays for  $X$ :

p,q	0				1				2			
0	(4.9,3.9)	(5.9,3.1)	(2.2,5.1)	(5.9,1.5)	(8.3,2.3)	(7.3,7.3)	(4.6,5.9)	(3.3,3.0)	(1.0,1.0)	(2.2,8.2)	(5.4,3.6)	(4.8,4.0)
	(6.8,4.6)	(6.7,9.8)	(2.1,0.5)	(3.5,2.9)	(4.5,8.9)	(0.2,0.9)	(3.6,5.0)	(3.4,0.4)	(8.8,1.0)	(8.9,5.8)	(2.0,2.6)	(5.2,9.5)
	(4.9,1.9)	(1.6,4.9)	(7.0,6.8)	(6.4,1.1)	(1.8,6.5)	(2.5,1.7)	(8.5,9.3)	(0.2,1.7)	(3.3,0.0)	(8.9,7.5)	(3.1,9.7)	(5.4,7.5)
	(2.9,7.6)	(7.5,5.5)	(0.6,4.6)	(9.0,7.6)	(6.4,5.2)	(7.8,5.3)	(8.7,9.1)	(9.1,5.5)	(8.7,8.3)	(6.3,5.4)	(1.5,9.6)	(4.4,4.4)

**Output:** Global matrix  $Y$ :

	Plane 0:	Plane 1:
B,D	0	
0	$\begin{bmatrix} (29.8,29.7) & (-1.9,1.1) & (-3.0,0.9) & (-3.0,-3.8) \\ (-3.3,1.0) & (0.4,-2.5) & (4.1,-3.9) & (-4.6,-1.4) \\ (-1.7,-1.2) & (0.1,3.4) & (0.9,5.0) & (-0.6,1.6) \\ (2.3,-0.5) & (1.0,-4.6) & (3.1,0.8) & (1.7,0.4) \\ (3.0,1.9) & (4.5,0.6) & (0.5,-3.8) & (-3.0,-0.1) \\ (1.0,0.1) & (-5.7,-1.9) & (-1.4,-1.2) & (0.8,2.6) \end{bmatrix}$	

Plane 2:		Plane 3:	
B,D	1		
0	$\begin{bmatrix} (-2.4,-2.8) & (2.0,0.1) \\ (2.1,-3.5) & (2.8,1.3) \\ (-1.7,0.7) & (2.8,1.0) \\ (-3.3,-2.2) & (-3.2,-5.1) \\ (-1.5,-3.1) & (1.4,0.1) \\ (1.8,0.6) & (-0.7,3.2) \end{bmatrix}$		$\begin{bmatrix} (3.6,-3.4) & (1.4,0.0) \\ (-1.9,-0.1) & (2.3,6.3) \\ (2.0,1.4) & (-0.6,1.0) \\ (-0.3,3.3) & (0.8,0.5) \\ (-1.3,-1.7) & (-2.7,0.7) \\ (0.2,2.9) & (1.0,-2.1) \end{bmatrix}$

The following is the  $1 \times 3$  process grid:

B,D	0	1	2
0	$P_{00}$	$P_{01}$	$P_{02}$

Local arrays for Y:

p,q	0				1			
0	(29.8,29.7)	(-1.9,1.1)	(-3.0,0.9)	(-3.0,-3.8)	(-2.4,-2.8)	(2.0,0.1)	(3.6,-3.4)	(1.4,0.0)
	(-3.3,1.0)	(0.4,-2.5)	(4.1,-3.9)	(-4.6,-1.4)	(2.1,-3.5)	(2.8,1.3)	(-1.9,-0.1)	(2.3,6.3)
	(-1.7,-1.2)	(0.1,3.4)	(0.9,5.0)	(-0.6,1.6)	(-1.7,0.7)	(2.8,1.0)	(2.0,1.4)	(-0.6,1.0)
	(2.3,-0.5)	(1.0,-4.6)	(3.1,0.8)	(1.7,0.4)	(-3.3,-2.2)	(-3.2,-5.1)	(-0.3,3.3)	(0.8,0.5)
	(3.0,1.9)	(4.5,0.6)	(0.5,-3.8)	(-3.0,-0.1)	(-1.5,-3.1)	(1.4,0.1)	(-1.3,-1.7)	(-2.7,0.7)
	(1.0,0.1)	(-5.7,-1.9)	(-1.4,-1.2)	(0.8,2.6)	(1.8,0.6)	(-0.7,3.2)	(0.2,2.9)	(1.0,-2.1)

There is not any data located on  $P_{02}$ .

## PSRCFT3 and PDRCFT3 — Real-to-Complex Fourier Transforms in Three Dimensions

### Purpose

These subroutines compute the mixed-radix three-dimensional complex conjugate even discrete Fourier transform of real data:

$$y_{k1,k2,k3} = scale \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \sum_{j3=0}^{n3-1} x_{j1,j2,j3} W_{n1}^{(Isign)j1k1} W_{n2}^{(Isign)j2k2} W_{n3}^{(Isign)j3k3}$$

for:

- $k1 = 0, 1, \dots, n1-1$
- $k2 = 0, 1, \dots, n2-1$
- $k3 = 0, 1, \dots, n3-1$

where:

$$W_{n1} = e^{-2\pi(\sqrt{-1})/n1}$$

$$W_{n2} = e^{-2\pi(\sqrt{-1})/n2}$$

$$W_{n3} = e^{-2\pi(\sqrt{-1})/n3}$$

and where:

- $x_{j1,j2,j3}$  are elements of array  $X$ .
- $y_{k1,k2,k3}$  are elements of array  $Y$ .
- $Isign$  is + or – (determined by argument  $isign$ ).
- $scale$  is a scalar value.

See references [1] and [3].

Table 128. Data Types

$X, scale$	$Y$	Subroutine
Short-precision real	Short-precision complex	PSRCFT3
Long-precision real	Long-precision complex	PDRCFT3

### Syntax

<b>Fortran</b>	CALL PSRCFT3   PDRCFT3 ( $x, y, n1, n2, n3, isign, scale, icontxt, ip$ )
<b>C and C++</b>	psrcft3   pdrctf3 ( $x, y, n1, n2, n3, isign, scale, icontxt, ip$ );

### On Entry

$x$  is the local array  $X$ , containing the three-dimensional data to be transformed that has been block-plane distributed over a  $1 \times q$  process grid, where  $q$  is the number of processes. (The value of  $ldx1$  and  $ldx2$  are set in the  $IP$  array.)

Scope: **local**

Specified as: an array of (at least) length  $ldx1 \times ldx2 \times LOCq(n3)$ , containing numbers of the data type indicated in Table 128 on page 825. This array must be aligned on a doubleword boundary.

*y* See On Return.

*n1* is the length of the first dimension of the three-dimensional data in the array to be transformed.

Scope: **global**

Specified as: a fullword integer;  $n1 \leq 37748736$  and must be one of the values listed in Figure 9 on page 798.

*n2* is the length of the second dimension of the three-dimensional data in the array to be transformed.

Scope: **global**

Specified as: a fullword integer;  $n2 \leq 37748736$  and must be one of the values listed in Figure 9 on page 798.

*n3* is the length of the third dimension of the three-dimensional data in the array to be transformed.

Scope: **global**

Specified as: a fullword integer;  $n3 \leq 37748736$  and must be one of the values listed in Figure 9 on page 798.

*isign* controls the direction of the transform, determining the sign, *isign*, of the exponent of  $W_m$ , where:

If *isign* = positive value,  $Isign = +$  (transforming time to frequency).

If *isign* = negative value,  $Isign = -$  (transforming frequency to time).

Scope: **global**

Specified as: a fullword integer; where  $isign > 0$  or  $isign < 0$ .

*scale* is the scaling constant *scale*.

Scope: **global**

Specified as: a number of the data type indicated in Table 128 on page 825, where  $scale > 0.0$  or  $scale < 0.0$ .

*icontxt* is the BLACS context parameter.

Scope: **global**

Specified as: the fullword integer that was returned by a prior call to BLACS\_GRIDINIT or BLACS\_GRIDMAP.

*ip* is an array of parameters,  $IP(i)$ , where:

- $IP(1)$  indicates whether the default values for *ip* are used or you set the values for *ip*.

If  $IP(1) = 0$ , then the following default values are used:

- $ldx1$  and  $ldx2$ , the leading dimensions of the array specified for  $X$ , equal  $n1$  and  $n2$ , respectively.
- $ldy1$  and  $ldy2$ , the leading dimensions of the array specified for  $Y$ , equal  $n3$  and  $n2$ , respectively.

The remaining parameters of the array  $IP$  are ignored.

If  $IP(1) \neq 0$ , then you set the remaining values of *ip* to indicate values for the leading dimensions.

- IP(2-19) are reserved.
- IP(20) indicates the value of the leading dimension,  $ldx1$ , of the array specified for  $X$ , where:  
If  $IP(20) = 0$ , then  $ldx1 = n1$ .  
If  $IP(20) \neq 0$ , then  $ldx1$  is this value of IP(20).
- IP(21) indicates the value of the leading dimension,  $ldx2$ , of the array specified for  $X$ , where:  
If  $IP(21) = 0$ , then  $ldx2 = n2$ .  
If  $IP(21) \neq 0$ , then  $ldx2$  is this value of IP(21).
- IP(22) indicates the value of the leading dimension,  $ldy1$ , of the array specified for  $Y$ , where:  
If  $IP(22) = 0$ , then  $ldy1 = n3$ .  
If  $IP(22) \neq 0$ , then  $ldy1$  is this value of IP(22).
- IP(23) indicates the value of the leading dimension,  $ldy2$ , of the array specified for  $Y$ , where:  
If  $IP(23) = 0$ , then  $ldy2 = n2$ .  
If  $IP(23) \neq 0$ , then  $ldy2$  is this value of IP(23).
- IP(24-40) are reserved.

Scope: **global**

Specified as: a one-dimensional array of (at least) length 40, containing fullword integers, where:

- IP(1) is any integer
- $IP(20) \geq n1$  or  $IP(20) = 0$
- $IP(21) \geq n2$  or  $IP(21) = 0$
- $IP(22) \geq n3$  or  $IP(22) = 0$
- $IP(23) \geq n2$  or  $IP(23) = 0$

### On Return

- $y$  is the local array  $Y$ , stored in FFT-packed storage mode, containing the results of the computation that are block-plane distributed, where:
- If  $IP(1) = 0$ , the local array  $Y$  has dimensions  $n3 \times n2 \times LOCq(n1/2)$ .
- If  $IP(1) \neq 0$ , the local array  $Y$  has dimensions  $ldy1 \times ldy2 \times LOCq(n1/2)$ .

Scope: **local**

Returned as: an  $ldy1 \times ldy2 \times LOCq(n1/2)$  array, containing the numbers of the data type indicated in Table 128 on page 825. This array must be aligned on a doubleword boundary.

### Notes and Coding Rules

1. These subroutines always return  $Y$  in transposed form.
2. For the output array  $Y$ , these subroutines may use any extra space available when  $ldy1$  and  $ldy2$  are greater than their minimum value.
3. You may specify the same array for  $X$  and  $Y$ . In this case, output overwrites input. If you specify different arrays  $X$  and  $Y$ , they must have no common elements; otherwise, results are unpredictable.
4. For more information on  $LOCq()$ , and how sequences are blocked-plane distributed and stored in FFT-packed storage mode, see "Three-Dimensional Sequences" on page 68.

In general, distributing your data evenly provides the best work load balance among the processes and allows the use of the most efficient collective communication. However, for your specific problem size and number of processes available, experimentation is necessary to achieve optimal performance.

## Error Conditions

### Computational Errors

None

### Resource Errors

1. Unable to allocate work space

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. *icontxt* is invalid

#### Stage 2:

1. Process grid is not  $1 \times q$
2. The subroutine was called from outside the process grid.

#### Stage 3:

1.  $n1 > 37748736$
2.  $n2 > 37748736$
3.  $n3 > 37748736$
4. The length of  $n1$ ,  $n2$ , or  $n3$  is not an allowable transform length.
5.  $isign = 0$
6.  $scale = 0.0$
7.  $IP(1) \neq 0$  and  $IP(20) \neq 0$  and  $IP(20) < n1$  (that is,  $ldx1 < n1$ )
8.  $IP(1) \neq 0$  and  $IP(21) \neq 0$  and  $IP(21) < n2$  (that is,  $ldx2 < n2$ )
9.  $IP(1) \neq 0$  and  $IP(22) \neq 0$  and  $IP(22) < n3$  (that is,  $ldy1 < n3$ )
10.  $IP(1) \neq 0$  and  $IP(23) \neq 0$  and  $IP(23) < n2$  (that is,  $ldy2 < n2$ )

## Examples

### Example

This example shows how to compute a three-dimensional transform. The data is block-plane distributed over a  $1 \times 2$  process grid. The arrays are declared as follows:

```
REAL*8 X(0:8,0:3,0:1)
COMPLEX*16 Y(0:4,0:3,0)
INTEGER*4 IP(40)
REAL*8 SCALE
```

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 1
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)
IP(1) = 1
IP(20) = 9
IP(21) = 4
IP(22) = 5
IP(23) = 4
```

```

          X   Y   N1  N2  N3  ISIGN  SCALE  ICONXT  IP
CALL PDRCFT3( X , Y , 4 , 4 , 4 , 1 , 1.000 , ICONXT , IP)

```

Global matrix  $X$ :

Plane 0:				Plane 1:				
B,D				0				
0	1.0 0.0 0.0 0.0				0.0 0.0 0.0 0.0			
	0.0 0.0 0.0 0.0				0.0 0.0 0.0 0.0			
	0.0 0.0 0.0 0.0				0.0 0.0 0.0 0.0			
	0.0 0.0 0.0 0.0				0.0 0.0 0.0 0.0			
	.				.			
	.				.			
	.				.			
	.				.			
	.				.			
	.				.			
Plane 2:				Plane 3:				
B,D				1				
0	0.0 0.0 0.0 0.0				0.0 0.0 0.0 0.0			
	0.0 0.0 0.0 0.0				0.0 0.0 0.0 0.0			
	0.0 0.0 0.0 0.0				0.0 0.0 0.0 0.0			
	0.0 0.0 0.0 0.0				0.0 0.0 0.0 0.0			
	.				.			
	.				.			
	.				.			
	.				.			
	.				.			
	.				.			

The following is the  $1 \times 2$  process grid:

B,D	0	1
0	$P_{00}$	$P_{01}$

Local arrays for  $X$ :

p,q	0								1							
0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

**Output:** The following global matrix  $Y$  is returned in transposed form and stored in FFT-packed storage mode:

Plane 0:

B,D	0
0	$\begin{bmatrix} (1.0,1.0) & (1.0,0.0) & (1.0,1.0) & (1.0,0.0) \end{bmatrix}$

## PSRCFT3 and PDRCFT3

$$0 \left[ \begin{array}{cccc} (1.0,0.0) & (1.0,0.0) & (1.0,0.0) & (1.0,0.0) \\ (1.0,1.0) & (1.0,0.0) & (1.0,1.0) & (1.0,0.0) \\ (1.0,0.0) & (1.0,0.0) & (1.0,0.0) & (1.0,0.0) \\ . & . & . & . \end{array} \right]$$

**Plane 1:**

$$\begin{array}{c} B,D \\ 0 \end{array} \begin{array}{c} 1 \\ \left[ \begin{array}{cccc} (1.0,0.0) & (1.0,0.0) & (1.0,0.0) & (1.0,0.0) \\ (1.0,0.0) & (1.0,0.0) & (1.0,0.0) & (1.0,0.0) \\ (1.0,0.0) & (1.0,0.0) & (1.0,0.0) & (1.0,0.0) \\ (1.0,0.0) & (1.0,0.0) & (1.0,0.0) & (1.0,0.0) \\ . & . & . & . \end{array} \right] \end{array}$$

The following is the  $1 \times 2$  process grid:

$$\begin{array}{c|c|c} B,D & 0 & 1 \\ \hline 0 & P_{00} & P_{01} \end{array}$$

The following local arrays for  $Y$  are returned in transposed form and stored in FFT-packed storage mode:

p,q	0				1			
0	(1.0,1.0)	(1.0,0.0)	(1.0,1.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)
	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)
	(1.0,1.0)	(1.0,0.0)	(1.0,1.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)
	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)
	.	.	.	.	.	.	.	.



## PSCRFT3 and PDCRFT3 — Complex-to-Real Fourier Transforms in Three Dimensions

### Purpose

These subroutines compute the mixed-radix three-dimensional real discrete Fourier transform of complex conjugate even data:

$$y_{k1,k2,k3} = scale \sum_{j1=0}^{n1-1} \sum_{j2=0}^{n2-1} \sum_{j3=0}^{n3-1} x_{j1,j2,j3} W_{n1}^{(Isign)j1k1} W_{n2}^{(Isign)j2k2} W_{n3}^{(Isign)j3k3}$$

for:

- $k1 = 0, 1, \dots, n1-1$
- $k2 = 0, 1, \dots, n2-1$
- $k3 = 0, 1, \dots, n3-1$

where:

$$W_{n1} = e^{-2\pi(\sqrt{-1})/n1}$$

$$W_{n2} = e^{-2\pi(\sqrt{-1})/n2}$$

$$W_{n3} = e^{-2\pi(\sqrt{-1})/n3}$$

and where:

- $x_{j1,j2,j3}$  are elements of array  $X$ .
- $y_{k1,k2,k3}$  are elements of array  $Y$ .
- $Isign$  is + or – (determined by argument  $isign$ ).
- $scale$  is a scalar value.

See references [1] and [3].

Table 129. Data Types

$X$	$Y, scale$	Subroutine
Short-precision complex	Short-precision real	PSCRFT3
Long-precision complex	Long-precision real	PDCRFT3

### Syntax

<b>Fortran</b>	CALL PSCRFT3   PDCRFT3 ( $x, y, n1, n2, n3, isign, scale, icontxt, ip$ )
<b>C and C++</b>	pscrft3   pdcrt3 ( $x, y, n1, n2, n3, isign, scale, icontxt, ip$ );

### On Entry

$x$  is the local array  $X$ , containing the three-dimensional data to be transformed that has been block-plane distributed over a  $1 \times q$  process grid, where  $q$  is the number of processes. (The value of  $ldx1$  and  $ldx2$  are set in the  $IP$  array.) Array  $X$  is stored in FFT-packed storage mode.

Scope: **local**

Specified as: an array of (at least) length  $ldx1 \times ldx2 \times LOCq(n1/2)$ , containing numbers of the data type indicated in Table 129 on page 831. This array must be aligned on a doubleword boundary.

*y* See On Return.

*n1* is the length of the first dimension of the three-dimensional data in the array to be transformed.

Scope: **global**

Specified as: a fullword integer;  $n1 \leq 37748736$  and must be one of the values listed in Figure 9 on page 798.

*n2* is the length of the second dimension of the three-dimensional data in the array to be transformed.

Scope: **global**

Specified as: a fullword integer;  $n2 \leq 37748736$  and must be one of the values listed in Figure 9 on page 798.

*n3* is the length of the third dimension of the three-dimensional data in the array to be transformed.

Scope: **global**

Specified as: a fullword integer;  $n3 \leq 37748736$  and must be one of the values listed in Figure 9 on page 798.

*isign* controls the direction of the transform, determining the sign, *isign*, of the exponent of  $W_m$ , where:

If *isign* = positive value,  $Isign = +$  (transforming time to frequency).

If *isign* = negative value,  $Isign = -$  (transforming frequency to time).

Scope: **global**

Specified as: a fullword integer; where  $isign > 0$  or  $isign < 0$ .

*scale* is the scaling constant *scale*.

Scope: **global**

Specified as: a number of the data type indicated in Table 129 on page 831, where  $scale > 0.0$  or  $scale < 0.0$ .

*icontxt* is the BLACS context parameter.

Scope: **global**

Specified as: the fullword integer that was returned by a prior call to BLACS\_GRIDINIT or BLACS\_GRIDMAP.

*ip* is an array of parameters,  $IP(i)$ , where:

- $IP(1)$  indicates whether the default values for *ip* are used or you set the values for *ip*.

If  $IP(1) = 0$ , then the following default values are used:

- $ldx1$  and  $ldx2$ , the leading dimensions of the array specified for  $X$ , equal  $n3$  and  $n2$ , respectively.
- $ldy1$  and  $ldy2$ , the leading dimensions of the array specified for  $Y$ , equal  $n1$  and  $n2$ , respectively.

The remaining parameters of the array *IP* are ignored.

If  $IP(1) \neq 0$ , then you set the remaining values of *ip* to indicate values for the leading dimensions.

- IP(2-19) are reserved.
- IP(20) indicates the value of the leading dimension,  $ldx1$ , of the array specified for X, where:  
If  $IP(20) = 0$ , then  $ldx1 = n3$ .  
If  $IP(20) \neq 0$ , then  $ldx1$  is this value of IP(20).
- IP(21) indicates the value of the leading dimension,  $ldx2$ , of the array specified for X, where:  
If  $IP(21) = 0$ , then  $ldx2 = n2$ .  
If  $IP(21) \neq 0$ , then  $ldx2$  is this value of IP(21).
- IP(22) indicates the value of the leading dimension,  $ldy1$ , of the array specified for Y, where:  
If  $IP(22) = 0$ , then  $ldy1 = n1$ .  
If  $IP(22) \neq 0$ , then  $ldy1$  is this value of IP(22).
- IP(23) indicates the value of the leading dimension,  $ldy2$ , of the array specified for Y, where:  
If  $IP(23) = 0$ , then  $ldy2 = n2$ .  
If  $IP(23) \neq 0$ , then  $ldy2$  is this value of IP(23).
- IP(24-40) are reserved.

Scope: **global**

Specified as: a one-dimensional array of (at least) length 40, containing fullword integers, where:

- IP(1) is any integer
- $IP(20) \geq n3$  or  $IP(20) = 0$
- $IP(21) \geq n2$  or  $IP(21) = 0$
- $IP(22) \geq n1$  or  $IP(22) = 0$
- $IP(23) \geq n2$  or  $IP(23) = 0$

## On Return

$y$  is the local array Y that is block-plane distributed and contains the results of the computation, where:

If  $IP(1) = 0$ , the local array Y has dimensions  $n1 \times n2 \times LOCq(n3)$ .

If  $IP(1) \neq 0$ , the local array Y has dimensions  $ldy1 \times ldy2 \times LOCq(n3)$ .

Scope: **local**

Returned as: an  $ldy1 \times ldy2 \times LOCq(n3)$  array, containing the numbers of the data type indicated in Table 129 on page 831. This array must be aligned on a doubleword boundary.

## Notes and Coding Rules

1. These subroutines always return Y in normal form.
2. For the output array Y, these subroutines may use any extra space available when  $ldy1$  and  $ldy2$  are greater than their minimum value.
3. You may specify the same array for X and Y. In this case, output overwrites input. If you specify different arrays X and Y, they must have no common elements; otherwise, results are unpredictable.
4. For more information on  $LOCq()$ , and how sequences are block-plane distributed and stored in FFT-packed storage mode, see "Three-Dimensional Sequences" on page 68.

In general, distributing your data evenly provides the best work load balance among the processes and allows the use of the most efficient collective communication. However, for your specific problem size and number of processes available, experimentation is necessary to achieve optimal performance.

## Error Conditions

### Computational Errors

None

### Resource Errors

1. Unable to allocate work space

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. *icontxt* is invalid

#### Stage 2:

1. Process grid is not  $1 \times q$
2. The subroutine was called from outside the process grid.

#### Stage 3:

1.  $n1 > 37748736$
2.  $n2 > 37748736$
3.  $n3 > 37748736$
4. The length of  $n1, n2, n3$  is not an allowable transform length.
5.  $isign = 0$
6.  $scale = 0.0$
7.  $IP(1) \neq 0$  and  $IP(20) \neq 0$  and  $IP(20) < n3$  (that is,  $ldx1 < n3$ )
8.  $IP(1) \neq 0$  and  $IP(21) \neq 0$  and  $IP(21) < n2$  (that is,  $ldx2 < n2$ )
9.  $IP(1) \neq 0$  and  $IP(22) \neq 0$  and  $IP(22) < n1$  (that is,  $ldy1 < n1$ )
10.  $IP(1) \neq 0$  and  $IP(23) \neq 0$  and  $IP(23) < n2$  (that is,  $ldy2 < n2$ )

## Examples

### Example

This example shows how to compute a three-dimensional transform. The data is block-plane distributed over a  $1 \times 2$  process grid. The arrays are declared as follows:

```
COMPLEX*16 X(0:4,0:3,0)
REAL*8 Y(0:8,0:3,0:1)
INTEGER*4 IP(40)
REAL*8 SCALE
```

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 1
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)
SCALE = 1.0D0/4*4*4
IP(1) = 1
IP(20) = 5
IP(21) = 4
IP(22) = 9
```

IP(23) = 4

```

          X   Y   N1  N2  N3  ISIGN      SCALE      ICONTXT  IP
CALL PDCRFT3( X , Y , 4 , 4 , 4 , -1 , 1.0D0/64.0D0 , ICONTXT , IP)

```

The following global matrix  $X$  is stored in FFT-packed storage mode:

**Plane 0:**

B,D 0

$$0 \begin{bmatrix} (1.0,1.0) & (1.0,0.0) & (1.0,1.0) & (1.0,0.0) \\ (1.0,0.0) & (1.0,0.0) & (1.0,0.0) & (1.0,0.0) \\ (1.0,1.0) & (1.0,0.0) & (1.0,1.0) & (1.0,0.0) \\ (1.0,0.0) & (1.0,0.0) & (1.0,0.0) & (1.0,0.0) \\ . & . & . & . \end{bmatrix}$$

**Plane 1:**

B,D 1

$$0 \begin{bmatrix} (1.0,0.0) & (1.0,0.0) & (1.0,0.0) & (1.0,0.0) \\ (1.0,0.0) & (1.0,0.0) & (1.0,0.0) & (1.0,0.0) \\ (1.0,0.0) & (1.0,0.0) & (1.0,0.0) & (1.0,0.0) \\ (1.0,0.0) & (1.0,0.0) & (1.0,0.0) & (1.0,0.0) \\ . & . & . & . \end{bmatrix}$$

The following is the  $1 \times 2$  process grid:

B,D	0	1
0	P <sub>00</sub>	P <sub>01</sub>

The following local arrays for  $X$  are stored in FFT-packed storage mode:

p,q	0				1			
0	(1.0,1.0)	(1.0,0.0)	(1.0,1.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)
	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)
	(1.0,1.0)	(1.0,0.0)	(1.0,1.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)
	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)	(1.0,0.0)
	.	.	.	.	.	.	.	.

**Output:**

Global matrix  $Y$ :

	Plane 0:	Plane 1:
B,D	0	
0	$\begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \end{bmatrix}$	$\begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \end{bmatrix}$
	Plane 2:	Plane 3:

## PSCRFT3 and PDCRFT3

B,D		1							
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.

The following is the  $1 \times 2$  process grid:

B,D	0	1
0	$P_{00}$	$P_{01}$

Local arrays for Y:

p,q	0								1							
0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

---

## Chapter 11. Random Number Generation

This chapter describes the random number generation subroutines.

---

### Overview of the Random Number Generation Subroutine

The random number generation subroutine generates uniformly distributed random numbers.

*Table 130. List of Random Number Generation Subroutines*

Descriptive Name	Long-Precision Subroutine	Page
Uniform Random Number Generator	PDURNG	839

---

## Random Number Generation Subroutine

This section contains the random number generation subroutine description.



## PDURNG — Uniform Random Number Generator

### Purpose

This subroutine generates a global vector  $x$  of  $n$  uniform pseudo-random numbers in the ranges (0,1) or (−1,1), depending on the *iopt* argument. The random numbers are generated using the multiplicative congruential method with a user-specified seed, as follows:

- $s_i = (a(s_{i-1})) \bmod (m) = (a^i s_0) \bmod (m)$
- $x_i = s_i/m$  if *iopt* = 0
- $x_i = (2s_i/m) - 1$  if *iopt* = 1
- for  $i = 1, 2, \dots, n$

where:

- $s_0$  is the initial seed provided by the caller.
- $s_i$  for  $i = 1, n$  is a random sequence.
- $x_i$  for  $i = 1, n$  are the random numbers.
- $a = 44485709377909.0$
- $m = 2.0^{48}$
- $n$  is the number of random numbers to be generated.

If  $n$  is 0, no computation is performed, and the initial seed is unchanged.

The global output vector  $x$  is distributed across the  $np$  processes, using block-cyclic distribution with a block size  $nb$ . (The processor grid can be one- or two-dimensions. For two dimensions, processes are selected in row-major order.) The length  $n$  of vector  $x$  must be a multiple of  $(np)(nb)$ .

Table 131. Data Types

$x, seed$	Subroutine
Long-precision real	PDURNG

### Syntax

<b>Fortran</b>	CALL PDURNG ( <i>seed</i> , <i>n</i> , <i>nb</i> , <i>x</i> , <i>iopt</i> , <i>icontxt</i> )
<b>C and C++</b>	pdurng ( <i>seed</i> , <i>n</i> , <i>nb</i> , <i>x</i> , <i>iopt</i> , <i>icontxt</i> );

### On Entry

*seed* is the initial value  $s_0$  used to generate the random numbers.

Scope: **global**

Specified as: a number of the data type indicated in Table 131. You should specify *seed* to be an **odd, whole** number; otherwise, PDURNG sets it to an odd, whole number and continues with the computation. The value of *seed* must be  $1.0 \leq seed < 2.0^{48}$ .

*n* is the global number of random numbers to be generated.

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$  and  $n$  must be divisible by  $(nb)(np)$ .

*nb* is the block size for vector  $x$ , used in the block-cyclic distribution.

Scope: **global**

Specified as: a fullword integer;  $nb > 0$ .

$x$  See On Return.

$iopt$  indicates the range of uniform random numbers to generate, where:  
 If  $iopt = 0$ , the range is (0,1).  
 If  $iopt = 1$ , the range is (-1,1).  
 Scope: **global**  
 Specified as: a fullword integer;  $iopt = 0$  or 1.

$icontxt$  is the BLACS context parameter.  
 Scope: **global**  
 Specified as: the fullword integer that was returned by a prior call to BLACS\_GRIDINIT or BLACS\_GRIDMAP.

### On Return

$seed$  is the new seed that is to be used to generate additional random numbers in subsequent invocations of PDURNG, having a value of  $seed = (a^n s_0) \bmod (m)$ .  
 Scope: **global**  
 Returned as: a number of the data type indicated in Table 131 on page 839. It is an **odd, whole** number, where  $1.0 \leq seed < 2.0^{48}$ .

$x$  is the local vector  $x$  of size  $n/np$ , containing the uniform pseudo-random numbers, where:  
 If  $iopt = 0$ , they are in the range (0,1).  
 If  $iopt = 1$ , they are in the range (-1,1).  
 Scope: **local**  
 Returned as: a one-dimensional array of (at least) length  $n/np$ , with a block size of  $nb$ , containing numbers of the data type indicated in Table 131 on page 839.

### Notes and Coding Rules

1. In your C program, argument  $seed$  must be passed by reference.
2. The suggested block size is (data cache size)/2, where, the data cache size can be obtained by utilizing the following C language code fragment:  

```
#include <sys/systemcfg.h>
int ics;
.
.
.
ics=_system_configuration.dcache_size/8;
```
3. There is no performance impact for  $nb = 1$ .
4. If you want  $n/np$  random numbers generated on each process, just set the block size to  $nb = n/np$ .
5. To generate more than  $(2^{31}-1)$  random numbers, you should make multiple calls to PDURNG.
6. The local vector  $x$  of length  $n/np$  can have sequential correlations. For details, see references [44], [50], [51], [52], and [53].

## Error Conditions

### Computational Errors

None

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. *icontxt* is invalid

#### Stage 2:

1. PDURNG was called from outside the process grid.

#### Stage 3:

1.  $seed < 1.0$  or  $seed \geq 2.0^{48}$
2.  $n < 0$
3.  $n$  is not divisible by  $(nb)(np)$
4.  $nb \leq 0$
5.  $iopt \neq 0$  or  $1$

## Examples

### Example

This example generates 30 random numbers in global vector  $x$  with a block size of 3, using block-cyclic distribution over a  $5 \times 1$  process grid.

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 5
NPCOL = 1
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)
```

```

          SEED  N      NB  X      IOPT  ICONTXT
          |    |      |   |      |      |
CALL PDURNG( SEED , 30 , 3 , X , 0 , ICONTXT)
```

```
SEED      = 31415926535897.0
```

**Note:** *icontxt* is the output of the BLACS\_GRIDINIT call.

**Output:**  $SEED = (a^{30}s_0) \bmod (m) = 6316434292705.0$

Global vector  $x$  with block size 3:

```
B,D      0
0  [ 0.683821516135299845
    0.058874407800946215
    0.391855250856924187
    -----
    0.755994653022330709
    0.557764301423606668
    0.001333801764989317
    -----
    0.056855932753212101
    0.331063036202269956
    0.347339794409027292
    -----
    0.649429020370863697 ]
```

3	0.386144876217390021
	0.457224855098420591
4	0.892518134165118937
	0.074548748224632532
	0.912379366805073033
5	0.112809499110515077
	0.857547605095465570
	0.756480901897081282
6	0.046993364463578046
	0.889457684002341153
	0.167775766106718294
7	0.504952722600595649
	0.999725924546471134
	0.696269487398215148
8	0.671896598019703362
	0.271472156040264423
	0.566418406688985243
9	0.464684865759100063
	0.982442539763031419
	0.022440482512937620

The following is the  $5 \times 1$  process grid:

B,D	0
0	$P_{00}$
5	
1	$P_{10}$
6	
2	$P_{20}$
7	
3	$P_{30}$
8	
4	$P_{40}$
9	

Local arrays for  $x$ :

p,q	0
0	0.683821516135299845
	0.058874407800946215
	0.391855250856924187
	0.112809499110515077
	0.857547605095465570
	0.756480901897081282
1	0.755994653022330709
	0.557764301423606668
	0.001333801764989317
	0.046993364463578046
	0.889457684002341153
	0.167775766106718294
	0.056855932753212101
	0.331063036202269956
	0.347339794409027292

2	0.504952722600595649
	0.999725924546471134
	0.696269487398215148
-----	-----
	0.649429020370863697
	0.386144876217390021
	0.457224855098420591
3	0.671896598019703362
	0.271472156040264423
	0.566418406688985243
-----	-----
	0.892518134165118937
	0.074548748224632532
	0.912379366805073033
4	0.464684865759100063
	0.982442539763031419
	0.022440482512937620

**PDURNG**

---

## Chapter 12. Utilities

This chapter describes the utility subroutines.

---

### Overview of the Utility Subroutines

The utility subroutines perform general service functions that support Parallel ESSL.

*Table 132. List of Utility Subroutines*

Descriptive Name	Subprogram	Page
Determine the Level of Parallel ESSL Installed on Your System	IPESL	847
Initialize a Type-1 Array Descriptor with Error Checking	DESCINIT	849
Initialize a Type-1 Array Descriptor	DESCSET	852
Compute the Ceiling of the Division of Two Integers	ICEIL	855
Compute the Least Common Multiple of Two Positive Integers	ILCM	856
Compute the Local Row or Column Index of a Global Element of a Block-Cyclically Distributed Matrix	INDXG2L	857
Compute the Process Row or Column Index of a Global Element of a Block-Cyclically Distributed Matrix	INDXG2P	859
Compute the Global Row or Column Index of a Local Element of a Block-Cyclically Distributed Matrix	INDXL2G	861
Compute the Starting Local Row or Column Index and Process Row or Column Index of a Global Element of a Block-Cyclically Distributed Matrix	INFOG1L	863
Compute the Starting Local Row and Column Indices and the Process Row and Column Indices of a Global Element of a Block-Cyclically Distributed Matrix	INFOG2L	865
Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process	NUMROC	868
General Matrix Norm	PDLANGE PZLANGE	872
Real Symmetric, Complex Symmetric, or Complex Hermitian Matrix Norm	PDLANSY PZLANSY PZLANHE	879

---

## Utility Subroutines

This section contains the utility subroutine descriptions.



## IPESSL — Determine the Level of Parallel ESSL Installed on Your System

### Purpose

This function returns the current level of Parallel ESSL installed on your system, where the level consists of a version number, release number, and modification number, plus the fix number of the most recent PTF installed.

**Note:** This subroutine is useful to you in those instances where your program is using a subroutine or feature that exists only in certain levels of Parallel ESSL. It is also useful when your program is dependent upon certain PTFs being applied to Parallel ESSL.

### Syntax

Fortran	IPESSL ()
C and C++	ipessl ();

### On Return

*Function value*

is the level of Parallel ESSL installed on your system. It is provided as a fullword integer in the form *vrrmmff*, where each two digits represents a part of the level:

- *vv* is the version number.
- *rr* is the release number.
- *mm* is the modification number.
- *ff* is the fix number of the most recent PTF installed.

Scope: **global**

Returned as: a fullword integer; *vrrmmff* > 0.

### Notes

1. To use IPESSL effectively, you must install your Parallel ESSL PTFs in their proper sequential order. As part of the result, IPESSL returns the value *ff* of the **most recent** PTF installed, rather than the **highest number** PTF installed. Therefore, if you do not install your PTFs sequentially, the *ff* value returned by IPESSL does not reflect the actual level of Parallel ESSL.
2. Declare the IPESSL function in your program as returning a fullword integer value.

### Examples

#### Example

This example shows several ways to use the IPESSL function. Most typically, you use IPESSL for checking the version and release level of Parallel ESSL. Suppose you are dependent on a new capability in Parallel ESSL, such as a new subroutine or feature, provided for the first time in (**fictitious**) Parallel ESSL Version 3 Release 2. You can add the following check in your program before using the new capability:

```
IF IPESSL() ≥ 3020000
```

## IPESL

By specifying 0000 for *mmff*, the modification and fix level, you are independent of the order in which your modifications and PTFs are installed.

Less typically, you use IPESL for checking the PTF level of Parallel ESSL. Suppose you are dependent on **(fictitious)** PTF 24 being installed on your Parallel ESSL Version 1 Release 0 system. You want to know whether to call a different user-callable subroutine to set up your array data. You can add the following check in your program before making the call:

```
IF IPESL() ≥ 1000024
```

If your system support group installed the Parallel ESSL PTFs in their proper sequential order, this test works properly; otherwise, it is unpredictable.

## DESCINIT — Initialize a Type-1 Array Descriptor with Error Checking

### Purpose

This subroutine initializes a type-1 array descriptor with error checking.

See reference [10].

### Syntax

<b>Fortran</b>	CALL DESCINIT ( <i>desc, m, n, mb, nb, irsrc, icsrc, ictxt, lld, info</i> )
<b>C and C++</b>	descinit ( <i>desc, m, n, mb, nb, irsrc, icsrc, ictxt, lld, info</i> );

### On Entry

*m* is the number of rows  $M_{\_}$  in a global matrix that has been block-cyclically distributed.

Scope: **global**

Specified as: a fullword integer;  $m \geq 0$ .

*n* is the number of columns  $N_{\_}$  in a global matrix that has been block-cyclically distributed.

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

*mb* is the row block size  $MB_{\_}$ .

Scope: **global**

Specified as: a fullword integer;  $mb > 0$ .

*nb* is the column block size  $NB_{\_}$ .

Scope: **global**

Specified as: a fullword integer;  $nb > 0$ .

*irsrc* is the process row  $RSRC_{\_}$  over which the first row of the global matrix is distributed.

Scope: **global**

Specified as: a fullword integer;  $0 \leq irsrc < p$ , where the process grid is  $p \times q$ .

*icsrc* is the process column  $CSRC_{\_}$  over which the first column of the global matrix is distributed.

Scope: **global**

Specified as: a fullword integer;  $0 \leq icsrc < q$ , where the process grid is  $p \times q$ .

*ictxt* is the BLACS context.

Scope: **global**

Specified as: a fullword integer;  $ictxt > 0$ .

*lld* is the leading dimension of the local array.

Scope: **local**

Specified as: a fullword integer;  $lld \geq \max(1, \text{LOCp}(m))$ .

### On Return

*desc* is the array descriptor for the global matrix, described in the following table:

<i>desc</i>	Name	Description	Value	Scope
1	DTYPE_	Descriptor type	DTYPE_ = 1	Global
2	CTXT_	BLACS context	CTXT_ = <i>ictxt</i>	Global
3	M_	Number of rows in the global matrix	M_ = $\max(m, 0)$	Global
4	N_	Number of columns in the global matrix	N_ = $\max(n, 0)$	Global
5	MB_	Row block size	MB_ = $\max(mb, 1)$	Global
6	NB_	Column block size	NB_ = $\max(nb, 1)$	Global
7	RSRC_	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	RSRC_ = $\max(0, \min(irsr, p-1))$	Global
8	CSRC_	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	CSRC_ = $\max(0, \min(icsrc, q-1))$	Global
9	LLD_	The leading dimension of the local array	LLD_ = $\max(lld, \max(1, \text{LOCp}(M_)))$	Local

Specified as: an array of (at least) length 9, containing fullword integers.

*info* has the following meaning:

If *info* = 0, the subroutine completed successfully.

If *info* < 0,  $-info$  was the first argument detected to have an invalid value.

Scope: **global**

Returned as: a fullword integer,  $info \leq 0$ .

## Error Conditions

### Computational Errors

None

### Resource Errors

None

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1. *ictxt* is invalid.

#### Stage 2:

1.  $m < 0$
2.  $n < 0$
3.  $mb < 1$
4.  $nb < 1$
5.  $irsr < 0$  or  $irsr \geq p$

6.  $icsrc < 0$  or  $icsrc \geq q$   
If  $mb \geq 1$ , then
7.  $lld < \max(1, LOC_p(m))$

## Examples

### Example

This example shows a call to DESCINIT to initialize a type-1 array descriptor for a global matrix of order 9 with  $4 \times 4$  blocking on a  $2 \times 2$  process grid.

**Note:** The example matrix is identical to the example matrix C used for NUMROC (see “Example” on page 869).

### Call Statements and Input:

```

NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

```

Calls to DESCINIT by process:

p,q	0	1
0	<pre> DESC  M  N  MB  NB  IRSRC  ICSRC  ICONTXT  LLD  INFO CALL DESCINIT(DESC, 9, 9, 4, 4, 0, 0, ICONTXT, 5, INFO) </pre>	<pre> DESC  M  N  MB  NB  IRSRC  ICSRC  ICONTXT  LLD  INFO CALL DESCINIT(DESC, 9, 9, 4, 4, 0, 0, ICONTXT, 5, INFO) </pre>
1	<pre> DESC  M  N  MB  NB  IRSRC  ICSRC  ICONTXT  LLD  INFO CALL DESCINIT(DESC, 9, 9, 4, 4, 0, 0, ICONTXT, 4, INFO) </pre>	<pre> DESC  M  N  MB  NB  IRSRC  ICSRC  ICONTXT  LLD  INFO CALL DESCINIT(DESC, 9, 9, 4, 4, 0, 0, ICONTXT, 4, INFO) </pre>

### Output:

The output returned by DESCINIT on each process:

	Process (0,0)	Process (0,1)	Process (1,0)	Process (1,1)
DESC(1)	1	1	1	1
DESC(2)	ICONTXT	ICONTXT	ICONTXT	ICONTXT
DESC(3)	9	9	9	9
DESC(4)	9	9	9	9
DESC(5)	4	4	4	4
DESC(6)	4	4	4	4
DESC(7)	0	0	0	0
DESC(8)	0	0	0	0
DESC(9)	5	5	4	4

The value of *info* is 0 on all processes.

## DESCSET — Initialize a Type-1 Array Descriptor

### Purpose

This subroutine initializes a type-1 array descriptor.

See reference [10].

### Syntax

<b>Fortran</b>	CALL DESCSET ( <i>desc, m, n, mb, nb, irsrc, icsrc, ictxt, lld</i> )
<b>C and C++</b>	descset ( <i>desc, m, n, mb, nb, irsrc, icsrc, ictxt, lld</i> );

### On Entry

*m* is the number of rows  $M_{\_}$  in a global matrix that has been block-cyclically distributed.

Scope: **global**

Specified as: a fullword integer;  $m \geq 0$ .

*n* is the number of columns  $N_{\_}$  in a global matrix that has been block-cyclically distributed.

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

*mb* is the row block size  $MB_{\_}$ .

Scope: **global**

Specified as: a fullword integer;  $mb > 0$ .

*nb* is the column block size  $NB_{\_}$ .

Scope: **global**

Specified as: a fullword integer;  $nb > 0$ .

*irsrc* is the process row  $RSRC_{\_}$  over which the first row of the global matrix is distributed.

Scope: **global**

Specified as: a fullword integer;  $0 \leq irsrc < p$ , where the process grid is  $p \times q$ .

*icsrc* is the process column  $CSRC_{\_}$  over which the first column of the global matrix is distributed.

Scope: **global**

Specified as: a fullword integer;  $0 \leq icsrc < q$ , where the process grid is  $p \times q$ .

*ictxt* is the BLACS context.

Scope: **global**

Specified as: a fullword integer;  $ictxt > 0$ .

*lld* is the leading dimension of the local array.

Scope: **local**

Specified as: a fullword integer;  $lld \geq \max(1, \text{LOCp}(M))$ .

### On Return

*desc* is the array descriptor for the global matrix, described in the following table:

<i>desc</i>	Name	Description	Value	Scope
1	DTYPE_	Descriptor type	DTYPE_ = 1	Global
2	CTXT_	BLACS context	CTXT_ = <i>ictxt</i>	Global
3	M_	Number of rows in the global matrix	M_ = $\max(m, 0)$	Global
4	N_	Number of columns in the global matrix	N_ = $\max(n, 0)$	Global
5	MB_	Row block size	MB_ = $\max(mb, 1)$	Global
6	NB_	Column block size	NB_ = $\max(nb, 1)$	Global
7	RSRC_	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	RSRC_ = $\max(0, \min(irsr, p-1))$	Global
8	CSRC_	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	CSRC_ = $\max(0, \min(icsrc, q-1))$	Global
9	LLD_	The leading dimension of the local array	LLD_ = $\max(lld, \max(1, \text{LOCp}(M)))$	Local

Specified as: an array of (at least) length 9, containing fullword integers.

## Examples

### Example

This example shows a call to DESCSET to initialize a type-1 array descriptor for a global matrix of order 9 with  $4 \times 4$  blocking on a  $2 \times 2$  process grid.

**Note:** The example matrix is identical to the example matrix C used for NUMROC (see “Example” on page 869).

#### Call Statements and Input:

```

NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

```

Calls to DESCSET by process:

## DESCSET

p,q	0	1
0	<pre> DESC  M  N  MB  NB  IRSRC  ICSRC  ICONTXT  LLD                             CALL DESCSET(DESC, 9, 9, 4, 4, 0, 0, ICONTXT, 5) </pre>	<pre> DESC  M  N  MB  NB  IRSRC  ICSRC  ICONTXT  LLD                             CALL DESCSET(DESC, 9, 9, 4, 4, 0, 0, ICONTXT, 5) </pre>
1	<pre> DESC  M  N  MB  NB  IRSRC  ICSRC  ICONTXT  LLD                             CALL DESCSET(DESC, 9, 9, 4, 4, 0, 0, ICONTXT, 4) </pre>	<pre> DESC  M  N  MB  NB  IRSRC  ICSRC  ICONTXT  LLD                             CALL DESCSET(DESC, 9, 9, 4, 4, 0, 0, ICONTXT, 4) </pre>

### Output:

The output returned by DESCSET on each process:

	Process (0,0)	Process (0,1)	Process (1,0)	Process (1,1)
DESC(1)	1	1	1	1
DESC(2)	ICONTXT	ICONTXT	ICONTXT	ICONTXT
DESC(3)	9	9	9	9
DESC(4)	9	9	9	9
DESC(5)	4	4	4	4
DESC(6)	4	4	4	4
DESC(7)	0	0	0	0
DESC(8)	0	0	0	0
DESC(9)	5	5	4	4



## ICEIL — Compute the Ceiling of the Division of Two Integers

### Purpose

This function returns the ceiling of the division of two integers.

See reference [10].

### Syntax

<b>Fortran</b>	ICEIL ( <i>inum</i> , <i>idenom</i> )
<b>C and C++</b>	ceil ( <i>inum</i> , <i>idenom</i> );

### On Entry

*inum* is the numerator.

Scope: **global**

Specified as: a fullword integer;  $inum \geq 0$ .

*idenom* is the denominator.

Scope: **global**

Specified as: a fullword integer;  $idenom > 0$ .

### On Return

*function value*  
is the ceiling function.

Scope: **global**

Returned as: a fullword integer.

### Examples

#### Example

This example shows an invocation of ICEIL to compute the ceiling of the division of two integers.

#### Invocation and Input:

```

              INUM IDENOM
              |    |
JCEIL  = ICEIL( 5,  3)

```

#### Output:

```
JCEIL  = 2
```

---

## ILCM — Compute the Least Common Multiple of Two Positive Integers

### Purpose

This function computes the least common multiple of two positive integers.

See reference [10].

### Syntax

<b>Fortran</b>	ILCM ( <i>i1</i> , <i>i2</i> )
<b>C and C++</b>	ilcm ( <i>i1</i> , <i>i2</i> );

### On Entry

*i1* is the first integer.

Scope: **global**

Specified as: a fullword integer;  $i1 > 0$ .

*i2* is the second integer.

Scope: **global**

Specified as: a fullword integer;  $i2 > 0$ .

### On Return

*function value*

is the integer value that is the least common multiple of *i1* and *i2*.

Scope: **global**

Returned as: a fullword integer;  $ilcm > 0$ .

### Examples

#### Example

This example shows an invocation of ILCM to compute the least common multiple of two positive integers.

**Invocation and Input:**

```

          I1 I2
          |  |
LCM  =  ILCM( 5, 3)

```

**Output:**

```
LCM  = 15
```

## INDXG2L — Compute the Local Row or Column Index of a Global Element of a Block-Cyclically Distributed Matrix

### Purpose

This function computes the local row or column of a global element of a block-cyclically distributed matrix.

See reference [10].

### Syntax

<b>Fortran</b>	INDXG2L ( <i>indxglob</i> , <i>nb</i> , <i>iproc</i> , <i>isrcproc</i> , <i>nprocs</i> )
<b>C and C++</b>	indxg2l ( <i>indxglob</i> , <i>nb</i> , <i>iproc</i> , <i>isrcproc</i> , <i>nprocs</i> );

### On Entry

*indxglob*

is the row or column index of a global matrix.

Scope: **global**

Specified as: a fullword integer; *indxglob* > 0.

*nb*

is the row block size MB\_ or the column block size NB\_.

Scope: **global**

Specified as: a fullword integer; *nb* > 0.

*iproc*

is ignored, but must be specified.

Scope: **global**

Specified as: a fullword integer.

*isrcproc*

is ignored, but must be specified.

Scope: **global**

Specified as: a fullword integer.

*nprocs*

is the number of rows *nprow* or the number of columns *npcol* in the process grid.

Scope: **global**

Specified as: a fullword integer; *nprocs* > 0.

### On Return

*function value*

is the local row or column index of the global matrix row or column identified by the global row or column index *indxglob*.

Scope: **global**

Returned as: a fullword integer; *indxg2l* > 0.

## Examples

### Example

This example shows the local invocations of INDXG2L from four processes in a  $2 \times 2$  process grid, using a global symmetric matrix of order 9.

**Note:** The example matrix is identical to the example matrix C used for NUMROC (see “Example” on page 869).

### Invocations and Input:

```
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONXTX)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)
```

```

      INDXGLOB NB IPROC ISRCPROC NPROCS
      |      |   |      |      |
IIC = INDXG2L( 6,  4,  0,      0,  NPROW)
```

```

      INDXGLOB NB IPROC ISRCPROC NPROCS
      |      |   |      |      |
JJC = INDXG2L( 9,  4,  0,      0,  NPCOL)
```

### Output:

```
IIC = 2
JJC = 5
```

## INDXG2P — Compute the Process Row or Column Index of a Global Element of a Block-Cyclically Distributed Matrix

### Purpose

This function computes the process row or column index of a global element of a block-cyclically distributed matrix.

See reference [10].

### Syntax

<b>Fortran</b>	INDXG2P ( <i>indxglob</i> , <i>nb</i> , <i>iproc</i> , <i>isrcproc</i> , <i>nprocs</i> )
<b>C and C++</b>	indxg2p ( <i>indxglob</i> , <i>nb</i> , <i>iproc</i> , <i>isrcproc</i> , <i>nprocs</i> );

### On Entry

*indxglob*

is the row or column index of a global matrix.

Scope: **global**

Specified as: a fullword integer; *indxglob* > 0.

*nb*

is the row block size MB\_ or the column block size NB\_.

Scope: **global**

Specified as: a fullword integer; *nb* > 0.

*iproc*

is ignored, but must be specified.

Scope: **global**

Specified as: a fullword integer.

*isrcproc*

is ignored, but must be specified.

Scope: **global**

Specified as: a fullword integer.

*nprocs*

is the number of rows *nprow* or the number of columns *npcol* in the process grid.

Scope: **global**

Specified as: a fullword integer; *nprocs* > 0.

### On Return

*function value*

is the index of the process row or column of the global matrix identified by the global row or column index *indxglob*.

Scope: **global**

Returned as: a fullword integer;  $0 \leq \text{indxg2p} < \text{nprocs}$ .

## Examples

### Example

This example shows the local invocations of INDXG2P from four processes in a  $2 \times 2$  process grid, using a global symmetric matrix of order 9.

**Note:** The example matrix is identical to the example matrix C used for NUMROC (see “Example” on page 869).

### Invocations and Input:

```

NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

```

```

                INDXGLOB NB IPROC ISRCPROC NPROCS
                |      |   |      |
ICROW = INDXG2P(  6,   4,  0,    0,  NPROW)

```

```

                INDXGLOB NB IPROC ISRCPROC NPROCS
                |      |   |      |
ICCOL = INDXG2P(  9,   4,  0,    0,  NPCOL)

```

### Output:

```

ICROW = 1
ICCOL = 0

```

## INDXL2G — Compute the Global Row or Column Index of a Local Element of a Block-Cyclically Distributed Matrix

### Purpose

This function computes the global row or column index of a local element of a block-cyclically distributed matrix.

See reference [10].

### Syntax

<b>Fortran</b>	INDXL2G ( <i>indxloc</i> , <i>nb</i> , <i>iproc</i> , <i>isrcproc</i> , <i>nprocs</i> )
<b>C and C++</b>	indxl2g ( <i>indxloc</i> , <i>nb</i> , <i>iproc</i> , <i>isrcproc</i> , <i>nprocs</i> );

### On Entry

*indxloc* is the local row or column index of the distributed matrix.

Scope: **local**

Specified as: a fullword integer;  $indxloc \geq 1$ .

*nb* is the row block size MB\_ or the column block size NB\_.

Scope: **global**

Specified as: a fullword integer;  $nb > 0$ .

*iproc* is the coordinate of the process from which the rows or columns of the local matrix are to be determined.

Scope: **local**

Specified as: a fullword integer;  $iproc \geq 0$ .

*isrcproc*

is the coordinate of the process that contains the first row or column of the local matrix.

Scope: **global**

Specified as: a fullword integer;  $isrcproc \geq 0$ .

*nprocs* is the number of processors over which the global matrix is distributed.

Scope: **global**

Specified as: a fullword integer;  $nprocs \geq 0$ .

### On Return

*function value*

is the global index of a distributed matrix entry referenced by the local index *indxloc* of the process indicated by *iproc*.

Scope: **global**

Returned as: a fullword integer.

## Examples

### Example

This example shows the local invocations of INDXL2G from four processes in a  $2 \times 2$  process grid, using a global symmetric matrix of order 9.

**Note:** The example matrix is identical to the example matrix C used for NUMROC (see “Example” on page 869).

#### Invocations and Input:

```

NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

```

#### Local invocations of INDXL2G:

p,q	0								1							
0	<div>INDXLOC NB IPROC ISRCPROC NPROCS</div>								<div>INDXLOC NB IPROC ISRCPROC NPROCS</div>							
	IC	=	INDXL2G(	1,	MB_C,	0,	RSRC_C,	NPROW)	IC	=	INDXL2G(	1,	MB_C,	0,	RSRC_C,	NPROW)
	<div>INDXLOC NB IPROC ISRCPROC NPROCS</div>								<div>INDXLOC NB IPROC ISRCPROC NPROCS</div>							
	JC	=	INDXL2G(	1,	NB_C,	0,	CSRC_C,	NPCOL)	JC	=	INDXL2G(	1,	NB_C,	1,	CSRC_C,	NPCOL)
1	<div>INDXLOC NB IPROC ISRCPROC NPROCS</div>								<div>INDXLOC NB IPROC ISRCPROC NPROCS</div>							
	IC	=	INDXL2G(	1,	MB_C,	1,	RSRC_C,	NPROW)	IC	=	INDXL2G(	1,	MB_C,	1,	RSRC_C,	NPROW)
	<div>INDXLOC NB IPROC ISRCPROC NPROCS</div>								<div>INDXLOC NB IPROC ISRCPROC NPROCS</div>							
	JC	=	INDXL2G(	1,	NB_C,	0,	CSRC_C,	NPCOL)	JC	=	INDXL2G(	1,	NB_C,	1,	CSRC_C,	NPCOL)

**Output:** The global row index IC and global column index JC returned by INDXL2G on each process:

p,q	0	1
0	IC=1	IC=1
	JC=1	JC=5
1	IC=5	IC=5
	JC=1	JC=5



## INFOG1L — Compute the Starting Local Row or Column Index and Process Row or Column Index of a Global Element of a Block-Cyclically Distributed Matrix

### Purpose

This subroutine computes the starting local row or column index and process row or column index of a global element of a block-cyclically distributed matrix.

See reference [10].

### Syntax

<b>Fortran</b>	CALL INFOG1L ( <i>gindx</i> , <i>nb</i> , <i>nprocs</i> , <i>myroc</i> , <i>isrcproc</i> , <i>lindx</i> , <i>rocsrc</i> )
<b>C and C++</b>	infog1l ( <i>gindx</i> , <i>nb</i> , <i>nprocs</i> , <i>myroc</i> , <i>isrcproc</i> , <i>lindx</i> , <i>rocsrc</i> );

### On Entry

*gindx* is the row or column starting index of a global submatrix.

Scope: **global**

Specified as: a fullword integer; *gindx* > 0.

*nb* is the row block size MB\_ or the column block size NB\_.

Scope: **global**

Specified as: a fullword integer; *nb* > 0.

*nprocs* is the number of rows or columns in the process grid.

Scope: **global**

Specified as: a fullword integer; *nprocs* > 0.

*myroc* is the process grid row or column index of the process calling this subroutine.

Scope: **local**

Specified as: a fullword integer; *myroc* ≥ 0.

*isrcproc*

is the process row or column over which the first row or column of the global matrix is distributed.

Scope: **global**

Specified as: a fullword integer; *isrcproc* ≥ 0.

### On Return

*lindx* is the starting local row or column index corresponding to the global submatrix row or column identified by *gindx*.

Scope: **local**

Returned as: a fullword integer; *lindx* ≥ 1.

**Note:** *lindx* is always at least 1 even if the calling process does not contain any pieces of the global submatrix row or column.

*rocsrc* is the process grid row or column index of the process that owns the first entry of the global submatrix row or column identified by *gindx*.

Scope: **global**

Returned as: a fullword integer;  $0 \leq \text{rocsrc} < \text{nprocs}$ .

## Examples

### Example

This example shows local calls to INFOG1L using a  $2 \times 2$  process grid and a global symmetric matrix of order 9.

**Note:** The example matrix is identical to the example matrix C used for NUMROC (see “Example” on page 869).

### Call Statements and Input:

```

NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)

```

### Local invocations of INFOG1L:

p,q	0							1						
	GINDX	NB	NPROCS	MYROC	ISRCPROC	LINDX	ROCSRC	GINDX	NB	NPROCS	MYROC	ISRCPROC	LINDX	ROCSRC
0	CALL INFOG1L(	5, MB_C,	NPROW,	0,	RSRC_C,	IIC,	ICROW)	CALL INFOG1L(	5, MB_C,	NPROW,	0,	RSRC_C,	IIC,	ICROW)
	CALL INFOG1L(	7, NB_C,	NPCOL,	0,	CSRC_C,	JJC,	ICCOL)	CALL INFOG1L(	7, NB_C,	NPCOL,	1,	CSRC_C,	JJC,	ICCOL)
1	CALL INFOG1L(	5, MB_C,	NPROW,	1,	RSRC_C,	IIC,	ICROW)	CALL INFOG1L(	5, MB_C,	NPROW,	1,	RSRC_C,	IIC,	ICROW)
	CALL INFOG1L(	7, NB_C,	NPCOL,	0,	CSRC_C,	JJC,	ICCOL)	CALL INFOG1L(	7, NB_C,	NPCOL,	1,	CSRC_C,	JJC,	ICCOL)

**Output:** The starting local row index IIC and column index JJC returned by INFOG1L on each process:

p,q	0	1
0	IIC=5 JJC=5	IIC=5 JJC=3
1	IIC=1 JJC=5	IIC=1 JJC=3

ICROW = 1 on all processes.

ICCOL = 1 on all processes.

## INFOG2L — Compute the Starting Local Row and Column Indices and the Process Row and Column Indices of a Global Element of a Block-Cyclically Distributed Matrix

### Purpose

This subroutine computes the starting local row and column indices and process row and column indices of a global element of a block-cyclically distributed matrix.

See reference [10].

### Syntax

<b>Fortran</b>	CALL INFOG2L ( <i>grindx</i> , <i>gcindx</i> , <i>desc</i> , <i>nprow</i> , <i>npcol</i> , <i>myrow</i> , <i>mycol</i> , <i>lrindx</i> , <i>lcindx</i> , <i>rsrc</i> , <i>csrc</i> )
<b>C and C++</b>	infog2l ( <i>grindx</i> , <i>gcindx</i> , <i>desc</i> , <i>nprow</i> , <i>npcol</i> , <i>myrow</i> , <i>mycol</i> , <i>lrindx</i> , <i>lcindx</i> , <i>rsrc</i> , <i>csrc</i> );

### On Entry

*grindx* is the global row starting index of the submatrix.

Scope: **global**

Specified as: a fullword integer;  $grindx \geq 1$ .

*gcindx* is the global column starting index of the submatrix.

Scope: **global**

Specified as: a fullword integer;  $gcindx \geq 1$ .

*desc* is the array descriptor for the global matrix, described in the following table:

<i>desc</i>	Name	Description	Limits	Scope
1	DTYPE_	Descriptor type	DTYPE_ = 1	Global
2	CTXT_	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_	Number of rows in the global matrix	$M_ \geq 0$	Global
4	N_	Number of columns in the global matrix	$N_ \geq 0$	Global
5	MB_	Row block size	$MB_ \geq 1$	Global
6	NB_	Column block size	$NB_ \geq 1$	Global
7	RSRC_	The process row over which the first row of the global matrix is distributed	$0 \leq RSRC_ < p$	Global
8	CSRC_	The process column which the first column of the global matrix is distributed	$0 \leq CSRC_ < q$	Global
9	LLD_	The leading dimension of the local array	$LLD_ \geq \max(1, LOCp(M_))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

*nproW* is the number of process rows over which the matrix is distributed.

Scope: **global**

Specified as: a fullword integer;  $nproW \geq 1$ .

*npcol* is the number of process columns over which the matrix is distributed.

Scope: **global**

Specified as: a fullword integer;  $npcol \geq 1$ .

*myrow* is the row coordinate of the process calling this subroutine.

Scope: **local**

Specified as: a fullword integer;  $0 \leq myrow < nproW$ .

*mycol* is the column coordinate of the process calling this subroutine.

Scope: **local**

Specified as: a fullword integer;  $0 \leq mycol < npcol$ .

### On Return

*lrindx* is the starting local row index of the submatrix.

Scope: **local**

Returned as: a fullword integer;  $lrindx \geq 1$ .

*lcindx* is the starting local column index of the submatrix.

Scope: **local**

Returned as: a fullword integer;  $lcindx \geq 1$ .

*rsrc* is the row coordinate of the process that contains the first row and column of the submatrix.

Scope: **global**

Returned as: a fullword integer;  $0 \leq rsrc < nproW$ .

*csrc* is the column coordinate of the process that contains the first row and column of the submatrix.

Scope: **global**

Returned as: a fullword integer;  $0 \leq csrc < npcol$ .

## Examples

### Example

This example demonstrates a call to INFOG2L using a  $2 \times 2$  process grid and a global symmetric matrix of order 9.

**Note:** The example matrix is identical to the example matrix C used for NUMROC (see “Example” on page 869).

### Call Statements and Input:

```
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)
```

p,q	0	1
0	<pre> GRINDX GCINDX  DESC  NPROW NPCOL                               CALL INFOG2L( 5,    7,  DESC_C,  2,    2, MYROW MYCOL  LRINDX LCINDX  RSRC   CSRC                               0,    0,    IIC,   JJC,  ICROW, ICCOL) </pre>	<pre> GRINDX GCINDX  DESC  NPROW NPCOL                               CALL INFOG2L( 5,    7,  DESC_C,  2,    2, MYROW MYCOL  LRINDX LCINDX  RSRC   CSRC                               0,    1,    IIC,   JJC,  ICROW, ICCOL) </pre>
1	<pre> GRINDX GCINDX  DESC  NPROW NPCOL                               CALL INFOG2L( 5,    7,  DESC_C,  2,    2, MYROW MYCOL  LRINDX LCINDX  RSRC   CSRC                               1,    0,    IIC,   JJC,  ICROW, ICCOL) </pre>	<pre> GRINDX GCINDX  DESC  NPROW NPCOL                               CALL INFOG2L( 5,    7,  DESC_C,  2,    2, MYROW MYCOL  LRINDX LCINDX  RSRC   CSRC                               1,    1,    IIC,   JJC,  ICROW, ICCOL) </pre>

**Output:** The starting local row index IIC and column index JJC returned by INFOG2L on each process:

p,q	0	1
0	IIC=5 JJC=5	IIC=5 JJC=3
1	IIC=1 JJC=5	IIC=1 JJC=3

ICROW = 1 on all processes.

ICCOL = 1 on all processes.

## NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process

### Purpose

This function computes the local number of rows or columns of a block-cyclically distributed matrix contained in a process row or process column, respectively, indicated by the calling sequence argument *iproc*.

See references [15] and [16].

### Syntax

<b>Fortran</b>	NUMROC ( <i>n</i> , <i>nb</i> , <i>iproc</i> , <i>isrcproc</i> , <i>nprocs</i> )
<b>C and C++</b>	numroc ( <i>n</i> , <i>nb</i> , <i>iproc</i> , <i>isrcproc</i> , <i>nprocs</i> );

### On Entry

*n* is the number of rows *M*<sub>–</sub> or columns *N*<sub>–</sub> in a global matrix that has been block-cyclically distributed.

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

*nb* is the row block size *MB*<sub>–</sub> or the column block size *NB*<sub>–</sub>.

Scope: **global**

Specified as: a fullword integer;  $nb > 0$ .

*iproc* is process row index *myrow* or the process column index *mycol*.

Scope: **local**

Specified as: a fullword integer;  $0 \leq iproc < nprocs$ .

*isrcproc* is the process row *RSRC*<sub>–</sub> or the process column *CSRC*<sub>–</sub> over which the first row or column, respectively, of the global matrix is distributed.

Scope: **global**

Specified as: a fullword integer;  $0 \leq isrcproc < nprocs$ .

*nprocs* is the number of rows *nprocw* or the number of columns *npcol* in the process grid.

Scope: **global**

Specified as: a fullword integer;  $nprocs > 0$ .

### On Return

*Function value*

is the local number of rows or columns of a block-cyclically distributed matrix contained in a process row or process column, respectively, indicated by the calling sequence argument *iproc*.

Scope: **local**

Returned as: a fullword integer.

## Notes

The variables  $p$  and  $nrow$  are used interchangeably to indicate the number of rows in a process grid. The variables  $q$  and  $ncol$  are used interchangeably to indicate the number of columns in a process grid.

## Error Conditions

### Computational Errors

None

### Resource Errors

None

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1.  $nb \leq 0$
2.  $nprocs \leq 0$

## Examples

### Example

This example shows the local invocations of NUMROC from four processes in a  $2 \times 2$  process grid, using a global symmetric matrix of order 9.

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)
```

The NUMROC function invocations and associated data on each process are shown in “Local Invocations of NUMROC.”

The array descriptor values for global matrix  $C$  are shown below:

	DESC_C
DTYPE_	1
CTXT_	<i>icontxt</i> <sup>1</sup>
M_	9
N_	9
MB_	4
NB_	4
RSRC_	0
CSRC_	0
LLD_	See below <sup>2</sup>

#### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:

## NUMROC

$LLD\_C = \text{MAX}(1, \text{NUMROC}(M\_C, MB\_C, MYROW, RSRC\_C, NPROW))$

In this example,  $LLD\_C = 5$  on  $P_{00}$  and  $P_{01}$ , and  $LLD\_C = 4$  on  $P_{10}$  and  $P_{11}$ .

Global symmetric matrix  $C$  of order 9 is stored in upper storage mode with block sizes  $4 \times 4$ :

B,D	0	1	2
0	$\begin{bmatrix} -6.0 & 0.0 & 0.0 & 0.0 \\ . & -6.0 & -2.0 & 0.0 \\ . & . & -6.0 & -2.0 \\ . & . & . & -6.0 \end{bmatrix}$	$\begin{bmatrix} 0.0 & -2.0 & -2.0 & 0.0 \\ -2.0 & -4.0 & 0.0 & -4.0 \\ -2.0 & 0.0 & 2.0 & 0.0 \\ 2.0 & 0.0 & 2.0 & 0.0 \end{bmatrix}$	$\begin{bmatrix} -2.0 \\ -2.0 \\ 6.0 \\ 2.0 \end{bmatrix}$
1	$\begin{bmatrix} . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \end{bmatrix}$	$\begin{bmatrix} -8.0 & -4.0 & 0.0 & -2.0 \\ . & -6.0 & 0.0 & -4.0 \\ . & . & -4.0 & 0.0 \\ . & . & . & -4.0 \end{bmatrix}$	$\begin{bmatrix} 0.0 \\ -6.0 \\ 0.0 \\ -4.0 \end{bmatrix}$
2	$\begin{bmatrix} . & . & . & . \end{bmatrix}$	$\begin{bmatrix} . & . & . & . \end{bmatrix}$	$\begin{bmatrix} -16.0 \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	$P_{00}$	$P_{01}$
2		
1	$P_{10}$	$P_{11}$

Local arrays for  $C$ :

p,q	0	1
0	$\begin{bmatrix} -6.0 & 0.0 & 0.0 & 0.0 & -2.0 \\ . & -6.0 & -2.0 & 0.0 & -2.0 \\ . & . & -6.0 & -2.0 & 6.0 \\ . & . & . & -6.0 & 2.0 \\ . & . & . & . & -16.0 \end{bmatrix}$	$\begin{bmatrix} 0.0 & -2.0 & -2.0 & 0.0 \\ -2.0 & -4.0 & 0.0 & -4.0 \\ -2.0 & 0.0 & 2.0 & 0.0 \\ 2.0 & 0.0 & 2.0 & 0.0 \\ . & . & . & . \end{bmatrix}$
1	$\begin{bmatrix} . & . & . & . & 0.0 \\ . & . & . & . & -6.0 \\ . & . & . & . & 0.0 \\ . & . & . & . & -4.0 \end{bmatrix}$	$\begin{bmatrix} -8.0 & -4.0 & 0.0 & -2.0 \\ . & -6.0 & 0.0 & -4.0 \\ . & . & -4.0 & 0.0 \\ . & . & . & -4.0 \end{bmatrix}$

Local Invocations of NUMROC:

p,q	0	1
0	$\begin{array}{l} \text{M\_C} \quad \text{MB\_C} \quad \text{MYROW} \quad \text{RSRC\_C} \quad \text{NPROW} \\ \text{MP} = \text{NUMROC}(9, 4, 0, 0, 2) \\ \text{N\_C} \quad \text{NB\_C} \quad \text{MYCOL} \quad \text{CSRC\_C} \quad \text{NPCOL} \\ \text{NQ} = \text{NUMROC}(9, 4, 0, 0, 2) \end{array}$	$\begin{array}{l} \text{M\_C} \quad \text{MB\_C} \quad \text{MYROW} \quad \text{RSRC\_C} \quad \text{NPROW} \\ \text{MP} = \text{NUMROC}(9, 4, 0, 0, 2) \\ \text{N\_C} \quad \text{NB\_C} \quad \text{MYCOL} \quad \text{CSRC\_C} \quad \text{NPCOL} \\ \text{NQ} = \text{NUMROC}(9, 4, 1, 0, 2) \end{array}$
1	$\begin{array}{l} \text{M\_C} \quad \text{MB\_C} \quad \text{MYROW} \quad \text{RSRC\_C} \quad \text{NPROW} \\ \text{MP} = \text{NUMROC}(9, 4, 1, 0, 2) \\ \text{N\_C} \quad \text{NB\_C} \quad \text{MYCOL} \quad \text{CSRC\_C} \quad \text{NPCOL} \\ \text{NQ} = \text{NUMROC}(9, 4, 0, 0, 2) \end{array}$	$\begin{array}{l} \text{M\_C} \quad \text{MB\_C} \quad \text{MYROW} \quad \text{RSRC\_C} \quad \text{NPROW} \\ \text{MP} = \text{NUMROC}(9, 4, 1, 0, 2) \\ \text{N\_C} \quad \text{NB\_C} \quad \text{MYCOL} \quad \text{CSRC\_C} \quad \text{NPCOL} \\ \text{NQ} = \text{NUMROC}(9, 4, 1, 0, 2) \end{array}$

Output:



The local number of rows MP and columns NQ of the block-cyclically distributed matrix returned by NUMROC on each process:

p,q	0	1
0	MP=5 NQ=5	MP=5 NQ=4
1	MP=4 NQ=5	MP=4 NQ=4

## PDLANGE and PZLANGE — General Matrix Norm

### Purpose

These functions compute the norm of real or complex general matrix  $A$ , where in this description  $A$  represents the global general submatrix  $A_{ia:ia+m-1, ja:ja+n-1}$ .

If  $m = 0$  or  $n = 0$ , then zero is returned as the value of the function.

Table 133. Data Types

$A$	$work$ , <b>Result</b>	<b>Subprogram</b>
Long-precision real	Long-precision real	PDLANGE
Long-precision complex	Long-precision real	PZLANGE

### Syntax

<b>Fortran</b>	PDLANGE   PZLANGE ( <i>norm</i> , <i>m</i> , <i>n</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>work</i> )
<b>C and C++</b>	pdlange   pzlange ( <i>norm</i> , <i>m</i> , <i>n</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>work</i> );

### On Entry

*norm* specifies the type of computation; where:

If *norm* = 'O' or '1', the one norm of  $A$  is computed.

If *norm* = 'T', the infinity norm of  $A$  is computed.

If *norm* = 'F' or 'E', the Frobenius norm of  $A$  is computed.

If *norm* = 'M', the absolute value of the matrix element having the largest absolute value, i.e.,  $\max(|A|)$ , is returned.

Scope: **global**

Specified as: a single character; *norm* = 'O', '1', 'T', 'F', 'E', or 'M'.

*m* is the number of rows in submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $m \geq 0$ .

*n* is the number of columns in submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

*a* is the local part of the global general matrix  $A$ . This identifies the **first element** of the local array  $A$ . This subroutine computes the location of the first element of the local subarray used, based on *ia*, *ja*, *desc\_a*, *p*, *q*, *myrow*, and *mycol*; therefore, the leading LOCp(*ia+m-1*) by LOCq(*ja+n-1*) part of the local array  $A$  must contain the local pieces of the leading *ia+m-1* by *ja+n-1* part of the global matrix.

Scope: **local**

Specified as: an LLD\_A by (at least) LOCq(N\_A) array, containing numbers of the data type indicated in Table 133. Details about the block-cyclic data distribution of global matrix  $A$  are stored in *desc\_a*.

*ia* is the row index of the global matrix *A*, identifying the first row of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq M\_A$  and  $ia+m-1 \leq M\_A$ .

*ja* is the column index of the global matrix *A*, identifying the first column of the submatrix *A*.

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq N\_A$  and  $ja+n-1 \leq N\_A$ .

*desc\_a* is the array descriptor for global matrix *A*, described in the following table:

<i>desc_a</i>	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	DTYPE_A=1	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $m = 0$ or $n = 0$ : $M\_A \geq 0$ Otherwise: $M\_A \geq 1$	Global
4	N_A	Number of columns in the global matrix	If $m = 0$ or $n = 0$ : $N\_A \geq 0$ Otherwise: $N\_A \geq 1$	Global
5	MB_A	Row block size	$MB\_A \geq 1$	Global
6	NB_A	Column block size	$NB\_A \geq 1$	Global
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq RSRC\_A < p$	Global
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq CSRC\_A < q$	Global
9	LLD_A	The leading dimension of the local array	$LLD\_A \geq \max(1, LOCp(M\_A))$	<b>Local</b>

Specified as: an array of (at least) length 9, containing fullword integers.

*work* a work array with at least the following dimension:

- 0, when *norm* = 'M', 'F', or 'E'; i.e., *work* is not used in the computation.
- *nq0*, when *norm* = 'O' or '1'
- *mp0*, when *norm* = 'T'

where:

- $iroffa = \text{mod}(ia-1, MB\_A)$
- $icoffa = \text{mod}(ja-1, NB\_A)$
- $iarow = \text{mod}(RSRC\_A + (ia-1)/MB\_A, nprow)$
- $iacol = \text{mod}(CSRC\_A + (ja-1)/NB\_A, npcyl)$
- $mp0 = \text{NUMROC}(m+iroffa, MB\_A, myrow, iarow, nprow)$
- $nq0 = \text{NUMROC}(n+icoffa, NB\_A, mycol, iacol, npcyl)$

Scope: **local**

Specified as: an area of storage containing numbers of data type indicated in Table 133 on page 872.

### On Return

*Function value*

If  $norm = 'O'$  or  $'1'$ , the one norm of  $A$  is returned.

If  $norm = 'I'$ , the infinity norm of  $A$  is returned.

If  $norm = 'F'$  or  $'E'$ , the Frobenius norm of  $A$  is returned.

If  $norm = 'M'$ , the absolute value of the matrix element having the largest absolute value, i.e.,  $\max(|A|)$ , is returned.

Scope: **global**

If  $m = 0$  or  $n = 0$ , the function returns zero.

Returned as: a long-precision real value.

**Note:** Declare this function in your program as returning a value of the data type indicated in Table 133 on page 872.

## Notes and Coding Rules

1. The NUMROC utility subroutine can be used to determine the values of  $LOCp(M\_)$  and  $LOCq(N\_)$  used in the argument descriptions above. For details, see “Determining the Number of Rows and Columns in Your Local Arrays” on page 28 and “NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process” on page 868.
2. This subprogram accepts lowercase letters for the  $norm$  argument.

## Error Conditions

### Computational Errors

None

### Resource Errors

None

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1.  $DTYPE\_A$  is invalid.

#### Stage 2:

1.  $CTXT\_A$  is invalid.

#### Stage 3:

1. This subroutine was called from outside the process grid.

#### Stage 4:

1.  $norm \neq 'O', '1', 'I', 'F', 'E',$  or  $'M'$
2.  $m < 0$
3.  $n < 0$
4.  $M\_A < 0$  and  $m = 0$  or  $n = 0$ ;  $M\_A < 1$  otherwise
5.  $N\_A < 0$  and  $m = 0$  or  $n = 0$ ;  $N\_A < 1$  otherwise
6.  $MB\_A < 1$
7.  $NB\_A < 1$

8.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
9.  $CSRC\_A < 0$  or  $CSRC\_A \geq q$
10.  $ia < 1$
11.  $ja < 1$

**Stage 5:**

If ( $m \neq 0$  and  $n \neq 0$ ):

1.  $ia > M\_A$
2.  $ja > N\_A$
3.  $ia+m-1 > M\_A$
4.  $ja+n-1 > N\_A$

**Stage 6:**

1.  $LLD\_A < \max(1, LOCp(M\_A))$

**Examples****Example 1**

This example computes the one norm of a real general matrix  $A$  of order 9, using a  $2 \times 2$  process grid.

**Call Statements and Input:**

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)
```

```

      NORM  M    N    A  IA  JA  DESC_A  WORK
      |    |    |    |  |  |  |        |
ANORM = PDLANGE('O', 9 , 9 , A , 1 , 1 , DESC_A , WORK )

```

	DESC_A
DTYPE_	1
CTXT_	<i>icontxt</i> <sup>1</sup>
M_	9
N_	9
MB_	3
NB_	3
RSRC_	1
CSRC_	0
LLD_	See below <sup>2</sup>

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:  
 $LLD\_A = \max(1, \text{NUMROC}(M\_A, MB\_A, MYROW, RSRC\_A, NPROW))$   
 In this example,  $LLD\_A = 3$  on  $P_{00}$  and  $P_{01}$ , and  $LLD\_A = 6$  on  $P_{10}$  and  $P_{11}$ .

**Input:**

## PDLANGE and PZLANGE

Global general  $9 \times 9$  matrix  $A$  with block size  $3 \times 3$ :

B,D	0	1	2
0	$\begin{bmatrix} 1.0 & 1.2 & 1.4 \\ 1.2 & 1.0 & 1.2 \\ 1.4 & 1.2 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 1.6 & 1.8 & 2.0 \\ 1.4 & 1.6 & 1.8 \\ 1.2 & 1.4 & 1.6 \end{bmatrix}$	$\begin{bmatrix} 2.2 & 2.4 & 2.6 \\ 2.0 & 2.2 & 2.4 \\ 1.8 & 2.0 & 2.2 \end{bmatrix}$
1	$\begin{bmatrix} 1.6 & 1.4 & 1.2 \\ 1.8 & 1.6 & 1.4 \\ 2.0 & 1.8 & 1.6 \end{bmatrix}$	$\begin{bmatrix} 1.0 & 1.2 & 1.4 \\ 1.2 & 1.0 & 1.2 \\ 1.4 & 1.2 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 1.6 & 1.8 & 2.0 \\ 1.4 & 1.6 & 1.8 \\ 1.2 & 1.4 & 1.6 \end{bmatrix}$
2	$\begin{bmatrix} 2.2 & 2.0 & 1.8 \\ 2.4 & 2.2 & 2.0 \\ 2.6 & 2.4 & 2.2 \end{bmatrix}$	$\begin{bmatrix} 1.6 & 1.4 & 1.2 \\ 1.8 & 1.6 & 1.4 \\ 2.0 & 1.8 & 1.6 \end{bmatrix}$	$\begin{bmatrix} 1.0 & 1.2 & 1.4 \\ 1.2 & 1.0 & 1.2 \\ 1.4 & 1.2 & 1.0 \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
1	$P_{00}$	$P_{01}$
0	$P_{10}$	$P_{11}$
2		

**Note:** The first row of  $A$  begins in the second row of the process grid.

Local arrays for  $A$ :

p,q	0	1
0	$\begin{bmatrix} 1.6 & 1.4 & 1.2 & 1.6 & 1.8 & 2.0 \\ 1.8 & 1.6 & 1.4 & 1.4 & 1.6 & 1.8 \\ 2.0 & 1.8 & 1.6 & 1.2 & 1.4 & 1.6 \end{bmatrix}$	$\begin{bmatrix} 1.0 & 1.2 & 1.4 \\ 1.2 & 1.0 & 1.2 \\ 1.4 & 1.2 & 1.0 \end{bmatrix}$
1	$\begin{bmatrix} 1.0 & 1.2 & 1.4 & 2.2 & 2.4 & 2.6 \\ 1.2 & 1.0 & 1.2 & 2.0 & 2.2 & 2.4 \\ 1.4 & 1.2 & 1.0 & 1.8 & 2.0 & 2.2 \\ 2.2 & 2.0 & 1.8 & 1.0 & 1.2 & 1.4 \\ 2.4 & 2.2 & 2.0 & 1.2 & 1.0 & 1.2 \\ 2.6 & 2.4 & 2.2 & 1.4 & 1.2 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 1.6 & 1.8 & 2.0 \\ 1.4 & 1.6 & 1.8 \\ 1.2 & 1.4 & 1.6 \\ 1.6 & 1.4 & 1.2 \\ 1.8 & 1.6 & 1.4 \\ 2.0 & 1.8 & 1.6 \end{bmatrix}$

**Output:**

$anorm = 16.2$  on all processes.

### Example 2

This example computes the one norm of a complex general matrix  $A$  of order 9, using a  $2 \times 2$  process grid.

**Call Statements and Input:**

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)
```

```

      NORM  M  N  A  IA  JA  DESC_A  WORK
      |    |  |  |  |  |  |  |
ANORM = PZLANGE('O', 9, 9, A, 1, 1, DESC_A, WORK)
```

	DESC_A
DTYPE_	1
CTXT_	<i>icontxt</i> <sup>1</sup>
M_	9
N_	9
MB_	3
NB_	3
RSRC_	1
CSRC_	0
LLD_	See below <sup>2</sup>

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:  
 $\text{LLD\_A} = \text{MAX}(1, \text{NUMROC}(\text{M\_A}, \text{MB\_A}, \text{MYROW}, \text{RSRC\_A}, \text{NPROW}))$   
In this example,  $\text{LLD\_A} = 3$  on  $P_{00}$  and  $P_{01}$ , and  $\text{LLD\_A} = 6$  on  $P_{10}$  and  $P_{11}$ .

Global general  $9 \times 9$  matrix  $A$  with block size  $3 \times 3$ :

B,D	0	1	2
0	(2.0, 1.0) (2.4,-1.0) (2.8,-1.0) (2.4, 1.0) (2.0, 1.0) (2.4,-1.0) (2.8, 1.0) (2.4, 1.0) (2.0, 1.0)	(3.2,-1.0) (3.6,-1.0) (4.0,-1.0) (2.8,-1.0) (3.2,-1.0) (3.6,-1.0) (2.4,-1.0) (2.8,-1.0) (3.2,-1.0)	(4.4,-1.0) (4.8,-1.0) (5.2,-1.0) (4.0,-1.0) (4.4,-1.0) (4.8,-1.0) (3.6,-1.0) (4.0,-1.0) (4.4,-1.0)
1	(3.2, 1.0) (2.8, 1.0) (2.4, 1.0) (3.6, 1.0) (3.2, 1.0) (2.8, 1.0) (4.0, 1.0) (3.6, 1.0) (3.2, 1.0)	(2.0, 1.0) (2.4,-1.0) (2.8,-1.0) (2.4, 1.0) (2.0, 1.0) (2.4,-1.0) (2.8, 1.0) (2.4, 1.0) (2.0, 1.0)	(3.2,-1.0) (3.6,-1.0) (4.0,-1.0) (2.8,-1.0) (3.2,-1.0) (3.6,-1.0) (2.4,-1.0) (2.8,-1.0) (3.2,-1.0)
2	(4.4, 1.0) (4.0, 1.0) (3.6, 1.0) (4.8, 1.0) (4.4, 1.0) (4.0, 1.0) (5.2, 1.0) (4.8, 1.0) (4.4, 1.0)	(3.2, 1.0) (2.8, 1.0) (2.4, 1.0) (3.6, 1.0) (3.2, 1.0) (2.8, 1.0) (4.0, 1.0) (3.6, 1.0) (3.2, 1.0)	(2.0, 1.0) (2.4,-1.0) (2.8,-1.0) (2.4, 1.0) (2.0, 1.0) (2.4,-1.0) (2.8, 1.0) (2.4, 1.0) (2.0, 1.0)

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
1	$P_{00}$	$P_{01}$
0	$P_{10}$	$P_{11}$
2		

**Note:** The first row of  $A$  begins in the second row of the process grid.

Local arrays for  $A$ :

p,q	0	1
0	(3.2, 1.0) (2.8, 1.0) (2.4, 1.0) (3.2,-1.0) (3.6,-1.0) (4.0,-1.0) (3.6, 1.0) (3.2, 1.0) (2.8, 1.0) (2.8,-1.0) (3.2,-1.0) (3.6,-1.0) (4.0, 1.0) (3.6, 1.0) (3.2, 1.0) (2.4,-1.0) (2.8,-1.0) (3.2,-1.0)	(2.0, 1.0) (2.4,-1.0) (2.8,-1.0) (2.4, 1.0) (2.0, 1.0) (2.4,-1.0) (2.8, 1.0) (2.4, 1.0) (2.0, 1.0)
1	(2.0, 1.0) (2.4,-1.0) (2.8,-1.0) (4.4,-1.0) (4.8,-1.0) (5.2,-1.0) (2.4, 1.0) (2.0, 1.0) (2.4,-1.0) (4.0,-1.0) (4.4,-1.0) (4.8,-1.0) (2.8, 1.0) (2.4, 1.0) (2.0, 1.0) (3.6,-1.0) (4.0,-1.0) (4.4,-1.0) (4.4, 1.0) (4.0, 1.0) (3.6, 1.0) (2.0, 1.0) (2.4,-1.0) (2.8,-1.0) (4.8, 1.0) (4.4, 1.0) (4.0, 1.0) (2.4, 1.0) (2.0, 1.0) (2.4,-1.0) (5.2, 1.0) (4.8, 1.0) (4.4, 1.0) (2.8, 1.0) (2.4, 1.0) (2.0, 1.0)	(3.2,-1.0) (3.6,-1.0) (4.0,-1.0) (2.8,-1.0) (3.2,-1.0) (3.6,-1.0) (2.4,-1.0) (2.8,-1.0) (3.2,-1.0) (3.2, 1.0) (2.8, 1.0) (2.4, 1.0) (3.6, 1.0) (3.2, 1.0) (2.8, 1.0) (4.0, 1.0) (3.6, 1.0) (3.2, 1.0)

**Output:**

## PDLANSY, PZLANSY, and PZLANHE

*anorm* = 33.73 on all processes.



## PDLANSY, PZLANSY, and PZLANHE — Real Symmetric, Complex Symmetric, or Complex Hermitian Matrix Norm

### Purpose

These functions compute the norm of real symmetric, complex symmetric, or complex Hermitian matrix  $A$ , where in this description  $A$  represents the global real symmetric, complex symmetric, or complex Hermitian submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$ .

If  $n = 0$ , then zero is returned as the value of the function.

Table 134. Data Types

$A$	<i>work</i> , <b>Result</b>	<b>Subprogram</b>
Long-precision real	Long-precision real	PDLANSY
Long-precision complex	Long-precision real	PZLANSY and PZLANHE

### Syntax

<b>Fortran</b>	PDLANSY   PZLANSY   PZLANHE ( <i>norm</i> , <i>uplo</i> , <i>n</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>work</i> )
<b>C and C++</b>	pdlansy   pzlansy   pzlanhe ( <i>norm</i> , <i>uplo</i> , <i>n</i> , <i>a</i> , <i>ia</i> , <i>ja</i> , <i>desc_a</i> , <i>work</i> );

### On Entry

*norm* specifies the type of computation; where:

If *norm* = 'O' or '1', the one norm of  $A$  is computed.

If *norm* = 'I', the infinity norm of  $A$  is computed.

If *norm* = 'F' or 'E', the Frobenius norm of  $A$  is computed.

If *norm* = 'M', the absolute value of the matrix element having the largest absolute value, i.e.,  $\max(|A|)$ , is returned.

Scope: **global**

Specified as: a single character; *norm* = 'O', '1', 'I', 'F', 'E', or 'M'.

*uplo* indicates whether the upper or lower triangular part of the global submatrix  $A$  is referenced, where:

If *uplo* = 'U', the upper triangular part is referenced.

If *uplo* = 'L', the lower triangular part is referenced.

Scope: **global**

Specified as: a single character; *uplo* = 'U' or 'L'.

*n* is the order of submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $n \geq 0$ .

*a* is the local part of the global real symmetric, complex symmetric, or complex Hermitian matrix  $A$ . This identifies the **first element** of the local array  $A$ . This subroutine computes the location of the first element of the local subarray used, based on *ia*, *ja*, *desc\_a*, *p*, *q*, *myrow*, and *mycol*;

therefore, the leading  $\text{LOCp}(ia+n-1)$  by  $\text{LOCq}(ja+n-1)$  part of the local array  $A$  must contain the local pieces of the leading  $ia+n-1$  by  $ja+n-1$  part of the global matrix, and:

- If  $uplo = 'U'$ , the leading  $n \times n$  upper triangular part of the global submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  must contain the upper triangular part of the submatrix, and the strictly lower triangular part is not referenced.
- If  $uplo = 'L'$ , the leading  $n \times n$  lower triangular part of the global submatrix  $A_{ia:ia+n-1, ja:ja+n-1}$  must contain the lower triangular part of the submatrix, and the strictly upper triangular part is not referenced.

Scope: **local**

Specified as: an  $\text{LLD\_A}$  by (at least)  $\text{LOCq}(\text{N\_A})$  array, containing numbers of the data type indicated in Table 134 on page 879. Details about the block-cyclic data distribution of global matrix  $A$  are stored in  $desc\_a$ .

$ia$  is the row index of the global matrix  $A$ , identifying the first row of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ia \leq \text{M\_A}$  and  $ia+n-1 \leq \text{M\_A}$ .

$ja$  is the column index of the global matrix  $A$ , identifying the first column of the submatrix  $A$ .

Scope: **global**

Specified as: a fullword integer;  $1 \leq ja \leq \text{N\_A}$  and  $ja+n-1 \leq \text{N\_A}$ .

$desc\_a$  is the array descriptor for global matrix  $A$ , described in the following table:

$desc\_a$	Name	Description	Limits	Scope
1	DTYPE_A	Descriptor type	$\text{DTYPE\_A} = 1$	Global
2	CTXT_A	BLACS context	Valid value, as returned by BLACS_GRIDINIT or BLACS_GRIDMAP	Global
3	M_A	Number of rows in the global matrix	If $n = 0$ : $\text{M\_A} \geq 0$ Otherwise: $\text{M\_A} \geq 1$	Global
4	N_A	Number of columns in the global matrix	If $n = 0$ : $\text{N\_A} \geq 0$ Otherwise: $\text{N\_A} \geq 1$	Global
5	MB_A	Row block size	$\text{MB\_A} \geq 1$	Global
6	NB_A	Column block size	$\text{NB\_A} \geq 1$	Global
7	RSRC_A	The process row of the $p \times q$ grid over which the first row of the global matrix is distributed	$0 \leq \text{RSRC\_A} < p$	Global
8	CSRC_A	The process column of the $p \times q$ grid over which the first column of the global matrix is distributed	$0 \leq \text{CSRC\_A} < q$	Global
9	LLD_A	The leading dimension of the local array	$\text{LLD\_A} \geq \max(1, \text{LOCp}(\text{M\_A}))$	Local

Specified as: an array of (at least) length 9, containing fullword integers.

- work* a work array with at least the following dimension:
- 0, when *norm* = 'M', 'F', or 'E'; that is, *work* is not used in the computation.
  - $2*nq0 + np0 + ldw$ , when *norm* = '1', 'O', or 'T'.

where:

- $iroffa = \text{mod}(ia-1, MB\_A)$
- $icoffa = \text{mod}(ja-1, NB\_A)$
- $iarow = \text{mod}(RSRC\_A + (ia-1) / MB\_A, nprow)$
- $iacol = \text{mod}(CSRC\_A + (ja-1) / NB\_A, npcol)$
- $np0 = \text{NUMROC}(n+iroffa, MB\_A, myrow, iarow, nprow)$
- $nq0 = \text{NUMROC}(n+icoffa, NB\_A, mycol, iacol, npcol)$
- $lcm = \text{ilcm}(nprow, npcol)$

where:

- *ldw* must have the following value:
  - If  $nprow \neq npcol$ ,  $ldw = MB\_A * \text{iceil}(\text{iceil}(np0 / MB\_A) / (lcm / nprow))$ .
  - If  $nprow = npcol$ ,  $ldw = 0$ .

Scope: **local**

Specified as: an area of storage containing numbers of data type indicated in Table 133 on page 872.

## On Return

*Function value*

If *norm* = 'O' or '1', the one norm of *A* is returned.

If *norm* = 'T', the infinity norm of *A* is returned.

If *norm* = 'F' or 'E', the Frobenius norm of *A* is returned.

If *norm* = 'M', the absolute value of the matrix element having the largest absolute value, i.e.,  $\max(|A|)$ , is returned.

Scope: **global**

If  $n = 0$ , the function returns zero.

Returned as: a long-precision real value.

**Note:** Declare this function in your program as returning a value of the data type indicated in Table 134 on page 879.

## Notes and Coding Rules

1. This subprogram accepts lowercase letters for the *norm* and *uplo* arguments.
2. On input to PZLANHE, the imaginary parts of the diagonal elements of the complex Hermitian matrix *A* are assumed to be zero, so you do not have to set these values.
3. *A* and *work* must have no common elements; otherwise, results are unpredictable.
4. The NUMROC utility subroutine can be used to determine the values of LOCp(M\_) and LOCq(N\_) used in the argument descriptions above. For details, see "Determining the Number of Rows and Columns in Your Local Arrays" on page 28 and "NUMROC — Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process" on page 868.

5. The global real symmetric, complex symmetric, or complex Hermitian matrix  $A$  must be distributed using a square block-cyclic distribution; that is,  $MB\_A = NB\_A$ .
6. The block row offset of  $A$  must be equal to the block column offset of  $A$ ; that is,  $\text{mod}(ia-1, MB\_A) = \text{mod}(ja-1, NB\_A)$ .

## Error Conditions

### Computational Errors

None

### Resource Errors

None

### Input-Argument and Miscellaneous Errors

#### Stage 1:

1.  $DTYPE\_A$  is invalid.

#### Stage 2:

1.  $CTXT\_A$  is invalid.

#### Stage 3:

1. This subroutine was called from outside the process grid.

#### Stage 4:

1.  $norm \neq 'O', 'I', 'T', 'F', 'E', \text{ or } 'M'$
2.  $uplo \neq 'U' \text{ or } 'L'$
3.  $n < 0$
4.  $M\_A < 0$  and  $n = 0$ ;  $M\_A < 1$  otherwise
5.  $N\_A < 0$  and  $n = 0$ ;  $N\_A < 1$  otherwise
6.  $ia < 1$
7.  $ja < 1$
8.  $MB\_A < 1$
9.  $NB\_A < 1$
10.  $RSRC\_A < 0$  or  $RSRC\_A \geq p$
11.  $CSRC\_A < 0$  or  $CSRC\_A \geq q$

#### Stage 5:

If  $n \neq 0$ :

1.  $ia > M\_A$
2.  $ja > N\_A$
3.  $ia+n-1 > M\_A$
4.  $ja+n-1 > N\_A$

#### Stage 6:

1.  $MB\_A \neq NB\_A$
2.  $\text{mod}(ia-1, MB\_A) \neq \text{mod}(ja-1, NB\_A)$

#### Stage 7:

1.  $LLD\_A < \max(1, \text{LOCp}(M\_A))$

## Examples

### Example 1

This example computes the infinity norm of a real symmetric matrix  $A$  of order 9, using a  $2 \times 2$  process grid.

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)
```

```

      NORM  UPLO  N   A  IA  JA  DESC_A  WORK
      |     |   |   |  |  |   |         |
ANORM = PDLANSY( 'I', 'L', 9, A, 1, 1, DESC_A, WORK )

```

	DESC_A
DTYPE_	1
CTXT_	<i>icontxt</i> <sup>1</sup>
M_	9
N_	9
MB_	3
NB_	3
RSRC_	0
CSRC_	0
LLD_	See below <sup>2</sup>

#### Notes:

- icontxt* is the output of the BLACS\_GRIDINIT call.
- Each process should set the LLD\_ as follows:  
 $\text{LLD\_A} = \text{MAX}(1, \text{NUMROC}(\text{M\_A}, \text{MB\_A}, \text{MYROW}, \text{RSRC\_A}, \text{NPROW}))$   
 In this example,  $\text{LLD\_A} = 6$  on  $P_{00}$  and  $P_{01}$ , and  $\text{LLD\_A} = 3$  on  $P_{10}$  and  $P_{11}$ .

#### Input:

Global real symmetric matrix  $A$  of order 9 with block size  $3 \times 3$ :

B,D	0	1	2
0	$\begin{bmatrix} 1.0 & . & . \\ 1.0 & 2.0 & . \\ 1.0 & 2.0 & 3.0 \end{bmatrix}$	$\begin{bmatrix} . & . & . \\ . & . & . \\ . & . & . \end{bmatrix}$	$\begin{bmatrix} . & . & . \\ . & . & . \\ . & . & . \end{bmatrix}$
1	$\begin{bmatrix} 1.0 & 2.0 & 3.0 \\ 1.0 & 2.0 & 3.0 \\ 1.0 & 2.0 & 3.0 \end{bmatrix}$	$\begin{bmatrix} 4.0 & . & . \\ 4.0 & 5.0 & . \\ 4.0 & 5.0 & 6.0 \end{bmatrix}$	$\begin{bmatrix} . & . & . \\ . & . & . \\ . & . & . \end{bmatrix}$
2	$\begin{bmatrix} 1.0 & 2.0 & 3.0 \\ 1.0 & 2.0 & 3.0 \\ 1.0 & 2.0 & 3.0 \end{bmatrix}$	$\begin{bmatrix} 4.0 & 5.0 & 6.0 \\ 4.0 & 5.0 & 6.0 \\ 4.0 & 5.0 & 6.0 \end{bmatrix}$	$\begin{bmatrix} 7.0 & . & . \\ 7.0 & 8.0 & . \\ 7.0 & 8.0 & 9.0 \end{bmatrix}$

The following is the  $2 \times 2$  process grid:

## PDLANSY, PZLANSY, and PZLANHE

B,D	0 2	1
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for A:

p,q	0						1		
0	1.0	.	.	.	.	.	.	.	.
	1.0	2.0	.	.	.	.	.	.	.
	1.0	2.0	3.0	.	.	.	.	.	.
	1.0	2.0	3.0	7.0	.	.	4.0	5.0	6.0
	1.0	2.0	3.0	7.0	8.0	.	4.0	5.0	6.0
	1.0	2.0	3.0	7.0	8.0	9.0	4.0	5.0	6.0
1	1.0	2.0	3.0	.	.	.	4.0	.	.
	1.0	2.0	3.0	.	.	.	4.0	5.0	.
	1.0	2.0	3.0	.	.	.	4.0	5.0	6.0

**Output:**

*anorm* = 45.0 on all processes.

### Example 2

This example computes the Frobenius norm of a complex symmetric matrix *A* of order 9, using a 2 × 2 process grid.

**Call Statements and Input:**

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)
```

```

          NORM  UPLO  N   A  IA  JA  DESC_A  WORK
          |     |     |   |   |   |     |   |
ANORM = PZLANSY( 'F', 'L', 9, A, 1, 1, DESC_A, WORK )
```

	DESC_A
DTYPE_	1
CTXT_	<i>icontxt</i> <sup>1</sup>
M_	9
N_	9
MB_	3
NB_	3
RSRC_	0
CSRC_	0
LLD_	See below <sup>2</sup>

**Notes:**

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:

LLD\_A = MAX(1, NUMROC(M\_A, MB\_A, MYROW, RSRC\_A, NPROW))

In this example, LLD\_A = 6 on P<sub>00</sub> and P<sub>01</sub>, and LLD\_A = 3 on P<sub>10</sub> and P<sub>11</sub>.

### Input:

Global complex symmetric matrix *A* of order 9 with block size  $3 \times 3$ :

B,D	0	1	2
0	$\begin{pmatrix} (18.0, 1.0) & \cdot & \cdot \\ (1.0, 1.0) & (18.0, 2.0) & \cdot \\ (1.0, 1.0) & (3.0, 1.0) & (18.0, 3.0) \end{pmatrix}$	$\begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix}$	$\begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix}$
1	$\begin{pmatrix} (1.0, 1.0) & (3.0, 1.0) & (5.0, 1.0) \\ (1.0, 1.0) & (3.0, 1.0) & (5.0, 1.0) \\ (1.0, 1.0) & (3.0, 1.0) & (5.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (18.0, 4.0) & \cdot & \cdot \\ (7.0, 1.0) & (18.0, 5.0) & \cdot \\ (7.0, 1.0) & (9.0, 1.0) & (18.0, 6.0) \end{pmatrix}$	$\begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix}$
2	$\begin{pmatrix} (1.0, 1.0) & (3.0, 1.0) & (5.0, 1.0) \\ (1.0, 1.0) & (3.0, 1.0) & (5.0, 1.0) \\ (1.0, 1.0) & (3.0, 1.0) & (5.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (7.0, 1.0) & (9.0, 1.0) & (11.0, 1.0) \\ (7.0, 1.0) & (9.0, 1.0) & (11.0, 1.0) \\ (7.0, 1.0) & (9.0, 1.0) & (11.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (18.0, 7.0) & \cdot & \cdot \\ (13.0, 1.0) & (18.0, 8.0) & \cdot \\ (13.0, 1.0) & (15.0, 1.0) & (18.0, 9.0) \end{pmatrix}$

The following is the  $2 \times 2$  process grid:

B,D	0 2	1
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for *A*:

p,q	0	1
0	$\begin{pmatrix} (18.0, 1.0) & \cdot & \cdot & \cdot & \cdot & \cdot \\ (1.0, 1.0) & (18.0, 2.0) & \cdot & \cdot & \cdot & \cdot \\ (1.0, 1.0) & (3.0, 1.0) & (18.0, 3.0) & \cdot & \cdot & \cdot \\ (1.0, 1.0) & (3.0, 1.0) & (5.0, 1.0) & (18.0, 7.0) & \cdot & \cdot \\ (1.0, 1.0) & (3.0, 1.0) & (5.0, 1.0) & (13.0, 1.0) & (18.0, 8.0) & \cdot \\ (1.0, 1.0) & (3.0, 1.0) & (5.0, 1.0) & (13.0, 1.0) & (15.0, 1.0) & (18.0, 9.0) \end{pmatrix}$	$\begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ (7.0, 1.0) & (9.0, 1.0) & (11.0, 1.0) \\ (7.0, 1.0) & (9.0, 1.0) & (11.0, 1.0) \\ (7.0, 1.0) & (9.0, 1.0) & (11.0, 1.0) \end{pmatrix}$
1	$\begin{pmatrix} (1.0, 1.0) & (3.0, 1.0) & (5.0, 1.0) & \cdot & \cdot & \cdot \\ (1.0, 1.0) & (3.0, 1.0) & (5.0, 1.0) & \cdot & \cdot & \cdot \\ (1.0, 1.0) & (3.0, 1.0) & (5.0, 1.0) & \cdot & \cdot & \cdot \end{pmatrix}$	$\begin{pmatrix} (18.0, 4.0) & \cdot & \cdot \\ (7.0, 1.0) & (18.0, 5.0) & \cdot \\ (7.0, 1.0) & (9.0, 1.0) & (18.0, 6.0) \end{pmatrix}$

### Output:

*anorm* = 81.88 on all processes.

### Example 3

This example computes the one norm of a complex Hermitian matrix *A* of order 9, using a  $2 \times 2$  process grid.

#### Call Statements and Input:

```
ORDER = 'R'
NPROW = 2
NPCOL = 2
CALL BLACS_GET(0, 0, ICONTXT)
CALL BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW, MYCOL)
```

```

      NORM  UPLO  N  A  IA  JA  DESC_A  WORK
      |    |    |  |  |  |    |    |
ANORM = PZLANHE( 'O', 'L', 9, A, 1, 1, DESC_A, WORK )
```

## PDLANSY, PZLANSY, and PZLANHE

	DESC_A
DTYPE_	1
CTXT_	<i>icontxt</i> <sup>1</sup>
M_	9
N_	9
MB_	3
NB_	3
RSRC_	0
CSRC_	0
LLD_	See below <sup>2</sup>

### Notes:

1. *icontxt* is the output of the BLACS\_GRIDINIT call.
2. Each process should set the LLD\_ as follows:  

$$\text{LLD\_A} = \text{MAX}(1, \text{NUMROC}(\text{M\_A}, \text{MB\_A}, \text{MYROW}, \text{RSRC\_A}, \text{NPROW}))$$
In this example, LLD\_A = 6 on P<sub>00</sub> and P<sub>01</sub>, and LLD\_A = 3 on P<sub>10</sub> and P<sub>11</sub>.

### Input:

Global complex Hermitian matrix *A* of order 9 with block size 3 × 3:

B,D	0	1	2
0	$\begin{pmatrix} (18.0, 0.0) & . & . \\ (1.0, 1.0) & (18.0, 0.0) & . \\ (1.0, 1.0) & (3.0, 1.0) & (18.0, 0.0) \end{pmatrix}$	$\begin{pmatrix} . & . & . \\ . & . & . \\ . & . & . \end{pmatrix}$	$\begin{pmatrix} . & . & . \\ . & . & . \\ . & . & . \end{pmatrix}$
1	$\begin{pmatrix} (1.0, 1.0) & (3.0, 1.0) & (5.0, 1.0) \\ (1.0, 1.0) & (3.0, 1.0) & (5.0, 1.0) \\ (1.0, 1.0) & (3.0, 1.0) & (5.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (18.0, 0.0) & . & . \\ (7.0, 1.0) & (18.0, 0.0) & . \\ (7.0, 1.0) & (9.0, 1.0) & (18.0, 0.0) \end{pmatrix}$	$\begin{pmatrix} . & . & . \\ . & . & . \\ . & . & . \end{pmatrix}$
2	$\begin{pmatrix} (1.0, 1.0) & (3.0, 1.0) & (5.0, 1.0) \\ (1.0, 1.0) & (3.0, 1.0) & (5.0, 1.0) \\ (1.0, 1.0) & (3.0, 1.0) & (5.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (7.0, 1.0) & (9.0, 1.0) & (11.0, 1.0) \\ (7.0, 1.0) & (9.0, 1.0) & (11.0, 1.0) \\ (7.0, 1.0) & (9.0, 1.0) & (11.0, 1.0) \end{pmatrix}$	$\begin{pmatrix} (18.0, 0.0) & . & . \\ (13.0, 1.0) & (18.0, 0.0) & . \\ (13.0, 1.0) & (15.0, 1.0) & (18.0, 0.0) \end{pmatrix}$

**Note:** On input, the imaginary parts of the diagonal elements of the complex Hermitian matrix *A* are assumed to be zero, so you do not have to set these values.

The following is the 2 × 2 process grid:

B,D	0 2	1
0	P <sub>00</sub>	P <sub>01</sub>
2		
1	P <sub>10</sub>	P <sub>11</sub>

Local arrays for *A*:



p,q	0						1		
0	(18.0, . )	.	.	.	.	.	.	.	.
	( 1.0, 1.0)	(18.0, . )	.	.	.	.	.	.	.
	( 1.0, 1.0)	( 3.0, 1.0)	(18.0, . )	.	.	.	.	.	.
	( 1.0, 1.0)	( 3.0, 1.0)	( 5.0, 1.0)	(18.0, . )	.	.	( 7.0, 1.0)	( 9.0, 1.0)	(11.0, 1.0)
	( 1.0, 1.0)	( 3.0, 1.0)	( 5.0, 1.0)	(13.0, 1.0)	(18.0, . )	.	( 7.0, 1.0)	( 9.0, 1.0)	(11.0, 1.0)
	( 1.0, 1.0)	( 3.0, 1.0)	( 5.0, 1.0)	(13.0, 1.0)	(15.0, 1.0)	(18.0, . )	( 7.0, 1.0)	( 9.0, 1.0)	(11.0, 1.0)
1	( 1.0, 1.0)	( 3.0, 1.0)	( 5.0, 1.0)	.	.	.	(18.0, . )	.	.
	( 1.0, 1.0)	( 3.0, 1.0)	( 5.0, 1.0)	.	.	.	( 7.0, 1.0)	(18.0, . )	.
	( 1.0, 1.0)	( 3.0, 1.0)	( 5.0, 1.0)	.	.	.	( 7.0, 1.0)	( 9.0, 1.0)	(18.0, . )

Output:

*anorm* = 82.92 on all processes.



---

## Part 3. Appendixes



---

## Appendix A. BLACS Quick Reference Guide

This quick reference guide provides information for using the BLACS subroutines, which are included with Parallel ESSL. It is divided into the following sections:

- “Calling sequences”
- “Argument data types” on page 894
- “Argument options” on page 894

### Notes:

1. To receive a complete copy of the *BLACS User's Guide*, send email to **netlib@ornl.gov** and type the following in the message:  
**send blacs\_ug.ps from blacs**
2. For additional information, see reference [55].

---

## Calling sequences

When calling one of the BLACS, you must pass each argument appropriately, as explained below:

**Pass by value:** All scalar arguments that are not modified (in other words, input-only arguments such as *incx*, *m*, and *lda*)

### Pass by reference:

The following types of arguments:

- arrays
- output arguments that are scalar data items (including all scalar data types such as real, complex, etc.)

There are two sets of calling sequences for the BLACS, as explained in the sections below:

- “Fortran interface for the BLACS”
- “C interface for the BLACS” on page 893

**Note:** In the calling sequences, an underlined argument indicates it is an output argument.

## Fortran interface for the BLACS

This section shows the calling sequences for the Fortran interface for the BLACS.

**Note:** If you are coding your program in C or C++, you may want to use the calling sequences shown in “C interface for the BLACS” on page 893. However, if you prefer to use the Fortran interface for the BLACS, you must pass the BLACS arguments by reference to the Parallel ESSL subroutine. See “Using the BLACS” on page 80 for sample C and C++ calling sequences that show arguments passed by reference.

### Initialization subroutines

CALL BLACS_PINFO ( <u>mypnum</u> , <u>nprocs</u> )
CALL BLACS_SETUP ( <u>mypnum</u> , <u>nprocs</u> )
CALL BLACS_GET ( <u>icontxt</u> , <u>what</u> , <u>val</u> )

CALL BLACS_SET ( <i>icontxt, what, val</i> )
CALL BLACS_GRIDINIT ( <i>icontxt, order, nprow, npcol</i> )
CALL BLACS_GRIDMAP ( <i>icontxt, usermap, ldumap, nprow, npcol</i> )

## Deallocating resources subroutines

CALL BLACS_FREEBUFF ( <i>icontxt, wait</i> )
CALL BLACS_GRIDEXIT ( <i>icontxt</i> )
CALL BLACS_ABORT ( <i>icontxt, errornum</i> )
CALL BLACS_EXIT ( <i>doneflag</i> )

## Sending subroutines

**Note:** In the subroutine names, GE indicates general rectangular matrix and TR indicates trapezoidal matrix.

CALL SGESD2D   DGESD2D   CGESD2D   ZGESD2D   IGESD2D ( <i>icontxt, m, n, a, lda, rdest, cdest</i> )
CALL SGEBS2D   DGEBS2D   CGEBS2D   ZGEBS2D   IGEBS2D ( <i>icontxt, scope, top, m, n, a, lda</i> )
CALL STRSD2D   DTRSD2D   CTRSD2D   ZTRSD2D   ITRSD2D ( <i>icontxt, uplo, diag, m, n, a, lda, rdest, cdest</i> )
CALL STRBS2D   DTRBS2D   CTRBS2D   ZTRBS2D   ITRBS2D ( <i>icontxt, scope, top, uplo, diag, m, n, a, lda</i> )

## Receiving subroutines

CALL SGERV2D   DGERV2D   CGERV2D   ZGERV2D   IGERV2D ( <i>icontxt, m, n, a, lda, rsrc, csrc</i> )
CALL SGEBR2D   DGEBR2D   CGEBR2D   ZGEBR2D   IGEBR2D ( <i>icontxt, scope, top, m, n, a, lda, rsrc, csrc</i> )
CALL STRRV2D   DTRRV2D   CTRRV2D   ZTRRV2D   ITRRV2D ( <i>icontxt, uplo, diag, m, n, a, lda, rsrc, csrc</i> )
CALL STRBR2D   DTRBR2D   CTRBR2D   ZTRBR2D   ITRBR2D ( <i>icontxt, scope, top, uplo, diag, m, n, a, lda, rsrc, csrc</i> )

## Global operation subroutines

Absolute Maximum Value of a General Matrix	CALL SGAMX2D   DGAMX2D   CGAMX2D   ZGAMX2D   IGAMX2D ( <i>icontxt, scope, top, m, n, a, lda, ra, ca, rflag, rdest, cdest</i> )
Absolute Minimum Value of a General Matrix	CALL SGAMN2D   DGAMN2D   CGAMN2D   ZGAMN2D   IGAMN2D ( <i>icontxt, scope, top, m, n, a, lda, ra, ca, rflag, rdest, cdest</i> )
Sum of the Elements of a General Matrix	CALL SGSUM2D   DGSUM2D   CGSUM2D   ZGSUM2D   IGSUM2D ( <i>icontxt, scope, top, m, n, a, lda, rdest, cdest</i> )

## Informational and miscellaneous subroutines

**Note:** BLACS\_PNUM returns an integer.

CALL BLACS_GRIDINFO ( <i>icontxt, nprow, npcol, myprow, mypcol</i> )
<i>ipnum</i> = BLACS_PNUM ( <i>icontxt, prow, pcol</i> )
CALL BLACS_PCOORD ( <i>icontxt, pnum, prow, pcol</i> )
CALL BLACS_BARRIER ( <i>icontxt, scope</i> )

## C interface for the BLACS

This section shows the calling sequences for the C interface for the BLACS.

### Initialization subroutines

void Cblacs_pinfo ( <i>mypnum, nprocs</i> )
void Cblacs_setup ( <i>mypnum, nprocs</i> )
void Cblacs_get ( <i>icontxt, what, val</i> )
void Cblacs_set ( <i>icontxt, what, val</i> )
void Cblacs_gridinit ( <i>icontxt, order, nprow, npcol</i> )
void Cblacs_gridmap ( <i>icontxt, usermap, ldumap, nprow, npcol</i> )

### Deallocating resources subroutines

void Cblacs_freebuff ( <i>icontxt, wait</i> )
void Cblacs_gridexit ( <i>icontxt</i> )
void Cblacs_abort ( <i>icontxt, errornum</i> )
void Cblacs_exit ( <i>doneflag</i> )

### Sending subroutines

**Note:** In the subroutine names, GE indicates general rectangular matrix and TR indicates trapezoidal matrix.

void Csgesd2d   Cdgcsd2d   Ccgcsd2d   Czgsd2d   Cigesd2d ( <i>icontxt, m, n, a, lda, rdest, cdest</i> )
void Csgebs2d   Cdgebs2d   Ccgebs2d   Czgebs2d   Cigebs2d ( <i>icontxt, scope, top, m, n, a, lda</i> )
void Cstrsd2d   Cdtrsd2d   Cctrsd2d   Cztrsd2d   Citrsd2d ( <i>icontxt, uplo, diag, m, n, a, lda, rdest, cdest</i> )
void Cstrbs2d   Cdtrbs2d   Cctrbs2d   Cztrbs2d   Citrbs2d ( <i>icontxt, scope, top, uplo, diag, m, n, a, lda</i> )

### Receiving subroutines

**Note:** In the subroutine names, GE indicates general rectangular matrix and TR indicates trapezoidal matrix.

void Csgerv2d   Cdgerv2d   Ccgerv2d   Czgerv2d   Cigerv2d ( <i>icontxt, m, n, a, lda, rsrc, csrc</i> )
void Csgebr2d   Cdgebr2d   Ccgebr2d   Czgebr2d   Cigebr2d ( <i>icontxt, scope, top, m, n, a, lda, rsrc, csrc</i> )
void Cstrrv2d   Cdtrrv2d   Cctrvv2d   Cztrrv2d   Citrvv2d ( <i>icontxt, uplo, diag, m, n, a, lda, rsrc, csrc</i> )
void Cstrbr2d   Cdtrbr2d   Cctrbr2d   Cztrbr2d   Citrbr2d ( <i>icontxt, scope, top, uplo, diag, m, n, a, lda, rsrc, csrc</i> )

### Global operation subroutines

Absolute Maximum Value of a General Matrix	void Csgamx2d   Cdgamx2d   Ccgamx2d   Czgamx2d   Cigamx2d ( <i>icontxt, scope, top, m, n, a, lda, ra, ca, rflag, rdest, cdest</i> )
Absolute Minimum Value of a General Matrix	void Csgamn2d   Cdgamn2d   Ccgamn2d   Czgamn2d   Cigamn2d ( <i>icontxt, scope, top, m, n, a, lda, ra, ca, rflag, rdest, cdest</i> )
Sum of the Elements of a General Matrix	void Csgsum2d   Cdgsum2d   Ccgsum2d   Czgsum2d   Cigsum2d ( <i>icontxt, scope, top, m, n, a, lda, rdest, cdest</i> )

## Informational and miscellaneous subroutines

**Note:** Cblacs\_pnum returns an integer.

void Cblacs_gridinfo ( <i>icontxt</i> , <i>nprow</i> , <i>npcol</i> , <i>mypro</i> , <i>mypcol</i> )
<i>ipnum</i> = Cblacs_pnum ( <i>icontxt</i> , <i>prow</i> , <i>pcol</i> )
void Cblacs_pcoord ( <i>icontxt</i> , <i>pnum</i> , <i>prow</i> , <i>pcol</i> )
void Cblacs_barrier ( <i>icontxt</i> , <i>scope</i> )

---

## Argument data types

This section shows the type of data for each BLACS argument.

Data Type	Argument
Character	<i>diag</i> , <i>order</i> , <i>scope</i> , <i>top</i> , <i>uplo</i>
Integer	A(LDA,*), BLACS_PNUM, CA(*), Cblacs_pnum, cdest, csrc, doneflag, errornum, icontxt, lda, ldumap, m, mypcol, mypnum, myprow, n, npcol, nprocs, nprow, pcol, pnum, prow, RA(*), rflag, rdest, rsrc, USERMAP(LDUMAP,NPCOL), VAL(*), wait, what
Short-precision real	A(LDA,*)
Long-precision real	A(LDA,*)
Short-precision complex	A(LDA,*)
Long-precision complex	A(LDA,*)

---

## Argument options

1. *order* = 'Row-major' or 'Column-major'  
The default is row-major natural ordering.
2. *uplo* = 'Upper trapezoidal' or 'Lower trapezoidal'
3. *diag* = 'Nonunit trapezoidal' or 'Unit trapezoidal'
4. *scope* = 'All', 'Row', or 'Column'
5. For broadcast topologies, *top* = '', 'T', 'D', 'H', 'S', 'F', 'M', 'T', '1', '2', '3', '4', '5', '6', '7', '8', or '9'  
For global topologies, *top* = '', 'H', 'F', 'T', '1', '2', '3', '4', '5', '6', '7', '8', or '9'

**Note:** In the MPI implementation of the BLACS for Parallel ESSL, the broadcast and global topology arguments are ignored. In all cases, the equivalent MPI collective communication subroutine is used.



---

## Appendix B. Sample Programs

This appendix contains information about sample programs provided with the Parallel ESSL product.

---

### Sample Programs and Utilities Provided with Parallel ESSL

A variety of sample programs are shipped with the Parallel ESSL product in the following directories:

**On AIX:**            /usr/lpp/pessl.rte.common/example/c  
                      /usr/lpp/pessl.rte.common/example/fortran

**On Linux:**        /opt/ibmmath/pessl/3.2/example/c  
                      /opt/ibmmath/pessl/3.2/example/fortran

For file names and installation procedures, see the *Parallel ESSL for AIX Installation Guide* or the *Parallel ESSL for Linux on pSeries Installation Guide*.

The sample programs include:

- A diffusion calculation example program, which is discussed at length later in this chapter. Two different versions of this sample program, in C and Fortran, are provided.
- Three sample programs, `image.f`, `pdgexmp.f`, and `simple.f`, plus utilities in a utility library. These are provided in the Fortran directory, along with a makefile. More information regarding these programs and utilities is provided in the following files:

**On AIX:**            /usr/lpp/pessl.rte.common/example/fortran/examples.readme

**On Linux:**        /opt/ibmmath/pessl/3.2/example/fortran/examples.readme

Following is a description of their functions:

- The image program demonstrates the use of the complex to real Fourier transform subroutines and calculates a correlation between two images.
- The `pdgexmp` program is a simple performance program, allowing you to vary the processor size and shape, as well as the block size, to gauge their effect on the performance of solutions to linear equations.
- The final program, `simple.f`, is an example program showing how to set up and use the sample utilities.
- The sample utility library consists of a set of Fortran 90 routines implementing commonly-used message passing functions. This example subroutine library provides you with a simplified interface to some very commonly-used message passing routines, including:
  - Library initialization:  
**initutils**  
BLACS initialization and initialization of library variables. This must be the first routine called in order to use the remainder of the library.
  - exitutils**  
Performs a BLACS grid exit and marks the library as uninitialized.
  - Scatter operation:  
**scatter** Scatter a matrix on a single node to a distributed matrix.

- Gather operation:  
**gather** Gather a matrix onto a single node from a distributed matrix.
- Nearest neighbor communication:
  - sendnorthborder**  
Sends top row of each block to north processor.
  - sendwestborder**  
Sends first column of each block to west processor.
  - sendsouthborder**  
Sends last row of each block to south processor.
  - sendeastborder**  
Sends last column of each block to east processor.
  - rcvnorthborder**  
Receives last row of each block from north processor.
  - rcvwestborder**  
Receives last column of each block from west processor.
  - rcvsouthborder**  
Receives top row of each block from south processor.
  - rcveastborder**  
Receives first column of each block from east processor.
- Array creation and descriptor vector initialization:
  - create\_array**  
Allocates memory for array and initializes array descriptor.
- Global-to-local mapping routines:
  - g2l** Creates global to local index arrays.
  - l2g** Creates local to global index arrays.
  - number\_row\_blocks**  
Returns number of local row blocks in an array.
  - number\_col\_blocks**  
Returns number of local column blocks in an array.
  - last\_row\_block\_size**  
Returns size of last local row block in an array.
  - last\_col\_block\_size**  
Returns size of last local column block an in array.
- Three sample programs using the sparse linear algebraic equations subroutines plus some utility programs. These programs are discussed in “Sample Sparse Linear Algebraic Equations Programs” on page 931. They are provided in the Fortran directory along with a makefile. More information regarding these programs and utilities is provided in the following files:
  - On AIX:**        /usr/lpp/pessl.rte.common/example/fortran/examples.readme
  - On Linux:**    /opt/ibmmath/pessl/3.2/example/fortran/examples.readme

---

## Sample Thermal Diffusion Program

A Fortran 90 sample program is presented in this section, along with the command for processing it in a parallel processing environment. The sample program, solving a thermal diffusion problem, uses vectors and matrices distributed across a one-dimensional process grid and call Level 2 PBLAS, Eigensystem analysis, and Fourier transform subroutines.

A copy of this sample program, plus an equivalent C program, is provided with the Parallel ESSL product. For file names and installation procedures, see the *Parallel ESSL Installation Memo*.

Following is a table of contents for this section, along with a description of each section for the Fortran 90 sample program.

*Table 135. Table of Contents for the Sample Thermal Diffusion Programs*

Contents	Page
<b>Thermal Diffusion Discussion Paper:</b> A technical description of the problem to be solved.	898
<b>Program main:</b> Finds the cooling rate for a specified set of points in an anisotropic rectangular beam, immersed in a constant heat bath with a temperature of zero.	902
<b>Module parameters:</b> Defines system wide parameters and the index structure used to help map global indices to local indices.	907
<b>Module diffusion:</b> Assigns problem parameters and initial data. <ul style="list-style-type: none"> <li>• <b>Subroutine init_diffusion:</b> Initializes the problem size, the number of output points, the functional form of the diffusion constant, and the initial temperature distribution.</li> <li>• <b>Subroutine init_temp:</b> Returns the initial temperature of the bar at a particular point.</li> <li>• <b>Subroutine diff_coef:</b> Returns the value of the thermal diffusion coefficient at an arbitrary point.</li> </ul>	908
<b>Module fourier:</b> Represents both the diffusion operator and the temperature profile in a sine function basis. <ul style="list-style-type: none"> <li>• <b>Subroutine get_diffusion_matrix:</b> Obtains the matrix representation of the diffusion operator in a sine function basis.</li> <li>• <b>Subroutine expand_temp_profile:</b> Obtains the expansion coefficients of the initial temperature profile in a sine function expansion.</li> <li>• <b>Subroutine factor_nodes:</b> Obtains the powers of prime factorization of the number of nodes, failing if the factorization is not compatible with FFT supported transform lengths.</li> <li>• <b>Subroutine min_power2:</b> Obtains the smallest number which is a power of 2 and greater than or equal to the input argument.</li> </ul>	911
<b>Module scale:</b> Initializes the communications and provides a few communication utility routines. <ul style="list-style-type: none"> <li>• <b>Subroutine initialize_scale:</b> Initializes BLACS and calculates a block size.</li> <li>• <b>Subroutine initialize_rarray:</b> Allocates space for a real array and creates the associated descriptor array and index array.</li> <li>• <b>Subroutine initialize_carray:</b> Allocates space for a complex array and creates the associated descriptor array and index array.</li> <li>• <b>Subroutine clocal_to_rglobal:</b> Gathers the real parts of the local portions of the block-cyclically-distributed complex array to generate the corresponding global matrix.</li> <li>• <b>Subroutine rlocal_to_rglobal:</b> Gathers the local portions of the block-cyclically-distributed real array to generate the corresponding global matrix.</li> </ul>	919
<b>Input Data:</b> Sample input data in namelist format, used by subroutine init_diffusion in module diffusion.	928
<b>Output Data:</b> Sample output data, based on the sample input data, issued at the end of program main.	928

Table 135. Table of Contents for the Sample Thermal Diffusion Programs (continued)

Contents	Page
<b>Makefile:</b> The makefile used to build the Fortran sample programs.	<b>For AIX</b> “32-Bit Makefile for Use with SMP Libraries on AIX” on page 967 “64-Bit Makefile for Use with SMP Libraries on AIX” on page 970 “32-Bit Makefile for Use with GM Libraries on AIX” on page 973 “64-Bit Makefile for Use with GM Libraries on AIX” on page 976  <b>For Linux</b> “32-Bit Makefile for Linux” on page 980 “64-Bit Makefile for Linux” on page 983
<b>Run Script:</b> The script file used to execute the Fortran sample programs.	<b>For AIX</b> “Run Script for Use with SMP Libraries on AIX” on page 972 “Run Script for Use with GM Libraries on AIX” on page 979  <b>For Linux</b> “Run Script for Linux” on page 985

## Thermal Diffusion Discussion Paper

The objective of the diffusion program is to solve for the temperature of a beam at any point and at any time, given an initial temperature distribution. The following assumptions are made concerning the beam and its properties:

- The beam has the dimensions  $l_x$ ,  $l_y$ , and  $l_z$ , where  $l_z \gg l_x$  and  $l_z \gg l_y$ .
- The thermal diffusion coefficient is a function of only:

$\bar{x}$  and  $\bar{y}$

That is, the physical properties of the beam do not depend on the direction:

$\bar{z}$

**Note:** The bar over a variable indicates it has dimension; whereas, the unbarred form is dimensionless.

- The beam is immersed in a zero degree heat bath.

The general diffusion equation is given by:

$$\frac{\partial}{\partial \bar{t}} T(\bar{x}, \bar{y}, \bar{z}, \bar{t}) = \nabla \cdot \bar{D} \nabla T(\bar{x}, \bar{y}, \bar{z}, \bar{t}) \quad (1)$$

where:

$$\bar{D}$$

is the position-dependent diffusion coefficient. Equation 1 may be rewritten, ignoring the  $z$  dimension, using the following dimensionless variables:

$$x = \frac{\pi \bar{x}}{l_x}$$

$$y = \frac{\pi \bar{y}}{l_y}$$

$$D = \frac{\bar{D}}{D_r}$$

$$t = \frac{l_x^2 \bar{t}}{\pi^2 D_r}$$

as follows:

$$\frac{\partial}{\partial t} T(x, y, t) = \left( \frac{\partial}{\partial x} D \frac{\partial}{\partial x} + \frac{\partial}{\partial y} l_r^2 D \frac{\partial}{\partial y} \right) T(x, y, t) \quad (2)$$

where  $D_r$  is a reference diffusion constant, and  $l_r$  is the beam dimension ratio  $l_x/l_y$ . This equation is subject to the initial and boundary conditions given by:

- $T(x, y, 0) = T_0(x, y)$
- $T(0, y, t) = T(x, 0, t) = T(\pi, y, t) = T(x, \pi, t) = 0$

In the program, the initial condition  $T_0(x, y)$ , the diffusion coefficient  $D(x, y)$ , and the ratio  $l_r$  are determined in the initialization subroutine **init\_diffusion**.  $n_x$  and  $n_y$ , defined later, are also initialized here. Equation 2 is solved by representing the operators in a sine function basis and solving the resulting matrix equations. We begin by expanding  $T$  in sine functions, as follows:

$$T(x, y, t) = \frac{2}{\pi} \sum_{k'=1}^{\infty} \sum_{j'=1}^{\infty} a_{k'j'}(t) \sin(k'x) \sin(j'y) \quad (3)$$

The initial set of expansion coefficients  $a_{kj}(0)$  are determined from the initial temperature profile by:

$$a_{kj}(0) = \frac{2}{\pi} \int_0^\pi \int_0^\pi \sin(kx) \sin(jy) T_0(x, y) dx dy \quad (4)$$

where the orthogonality of the sine functions has been used. If we extend the range of  $T$  from  $\pi$  to  $2\pi$ , as an odd function, equation 4 can be written in terms of a discrete Fourier transform:

$$a_{kj}(0) = \frac{-1}{2\pi} \int_0^{2\pi} \int_0^{2\pi} e^{-ikx-ijy} T_0(x, y) dx dy \quad (5)$$

$a_{kj}(0)$  is calculated in the program in the subroutine **expand\_temp\_profile**, with the actual temperatures calculated in the function **init\_temp**. The subroutine call to DCFT2 performs the Fourier transform, and the results are stored in the array A. Substitute equation 3 into equation 2, multiply by  $(2/\pi)\sin(kx)\sin(jy)$  and integrate over  $x$  and  $y$  to obtain:

$$\frac{\partial a_{kj}(t)}{\partial t} = - \sum_{k'=1}^{\infty} \sum_{j'=1}^{\infty} F_{kj;k'j'} a_{k'j'}(t) \quad (6)$$

where:

$$F_{kj;k'j'} = \frac{-4}{\pi^2} \int_0^\pi \int_0^\pi \sin(kx) \sin(jy) D_{op} \sin(k'x) \sin(j'y) dx dy \quad (7)$$

and:

$$D_{op} = \left( \frac{\partial}{\partial x} D \frac{\partial}{\partial x} + \frac{\partial}{\partial y} l_r^2 D \frac{\partial}{\partial y} \right) \quad (8)$$

Note that  $\sin(kx)\sin(k'x)$  and  $\sin(jy)\sin(j'y)$  are even functions about  $\pi$ . Therefore, if we define  $D(2\pi-x, y) = D(x, y)$  and  $D(x, 2\pi-y) = D(x, y)$ , where  $0 \leq x \leq \pi$  and  $0 \leq y \leq \pi$ , the limits on the integral in equation 7 may be extended to  $2\pi$ :

$$F_{kj;k'j'} = \frac{1}{\pi^2} \int_0^{2\pi} \int_0^{2\pi} \left( kk' D \cos(kx) \cos(k'x) \sin(jy) \sin(j'y) + l_r^2 jj' D \cos(jy) \cos(j'y) \sin(kx) \sin(k'x) \right) dx dy \quad (9)$$

Equation 9 may be further reduced to sums of Fourier transforms using the identities:

- $\cos(kx)\cos(k'x) = (1/2)(\cos((k-k')x) + \cos((k+k')x))$
- $\sin(kx)\sin(k'x) = (1/2)(\cos((k-k')x) - \cos((k+k')x))$
- $\sin(kx) = (i/2)(e^{ikx} - e^{-ikx})$
- $\cos(kx) = (1/2)(e^{ikx} + e^{-ikx})$

Substituting these identities into equation 9, we have:

$$F_{kj;k'j'} = (kk' + l_r^2 jj') (\tilde{D}_{k^- j^-} - \tilde{D}_{k^+ j^+}) + (kk' - l_r^2 jj') (\tilde{D}_{k^+ j^-} - \tilde{D}_{k^- j^+}) \quad (10)$$

where:

- $k^+ = k+k'$
- $k^- = |k-k'|$
- $j^+ = j+j'$
- $j^- = |j-j'|$

and where:

$$\tilde{D}_{kj} = \frac{1}{4\pi^2} \int_0^{2\pi} \int_0^{2\pi} D e^{-ikx} e^{-ijy} dx dy$$

Equation 10 is the simplest form for the matrix elements of the diffusion operator, and these elements are calculated in the subroutine **get\_diffusion\_matrix**. The diffusion coefficient is evaluated with the function **diff\_coef**. The Parallel ESSL Fourier transform subroutine PDCFT2, called in subroutine **get\_diffusion\_matrix**, is used to determine the following elements:

$$\tilde{D}_{kj}$$

which are stored in the array DF. Because DF is an array which is block cyclically distributed among the nodes and each node requires elements of DF not locally available, this array is collected to the global array DFG on each node. The array DFG is subsequently used to calculate the matrix  $F$ . Now that we have determined the matrix elements of equation 6, we must truncate it in order to solve it. This may be done by truncating the  $k$  summation at  $n_x$  and the  $j$  summation at  $n_y$ . The dual indices may be collapsed into a single index 1 by:

- $l = (j-1)n_x + k$

The  $j$  and  $k$  indices can similarly be recovered from the 1 index by:

- $j = ((l-1)/n_x) + 1$
- $k = \text{mod}(l-1, n_x) + 1$

Rewriting the truncated equation 6, we have:

$$\frac{\partial a_l(t)}{\partial t} = - \sum_{l'=1}^{n_x n_y} F_{l;l'} a_{l'}(t) \quad (11)$$

Equation 11 has the general solution:

$$a_l(t) = \sum_{l'=1}^{n_x n_y} e^{-\lambda_{l'} t} B_{ll'} \left[ \sum_{m=1} B_{ml'} a_m(0) \right] \quad (12)$$

where  $\lambda$  is the eigenvalue vector and  $B$  is the matrix of eigenvectors of the matrix  $F$ . The eigenvectors  $B$  and eigenvalues  $\lambda$  correspond to the arrays B and LAMBDA in

the program. These eigenvalues and eigenvectors are determined with the Parallel ESSL subroutine PDSYEVX. The inner sum:

$$\sum_{m=1} B_{ml} a_m(0)$$

corresponds to the array AB, with ABT containing the extra factor of:

$$e^{-\lambda_l t}$$

Also,  $a_l(t)$  is represented by the array X, which is reused for each solution time.

Each of these matrix multiplications are done using the Parallel ESSL subroutine PDGEMV. The final solution is obtained by summing over the expansion coefficients:

$$T(x, y, t) = \sum_{k=1}^{n_x} \sum_{j=1}^{n_y} \sin(kx) \sin(jy) a_{kj}(t) \quad (13)$$

The final temperature array is TEMP in the program, with the indices into the array corresponding to specific values of  $x$ ,  $y$ , and  $t$ .

## Program Main

```

      program main
!
! Purpose, to find the cooling rate for a specified set of
! points in an anisotropic rectangular beam, immersed in a constant
! heat bath with a temperature of 0.
!
! Routines called:
!   expand_temp_profile
!   get_diffusion_matrix
!   igamx2d
!   init_diffusion
!   initialize_rarray
!   initialize_scale
!   pdgemv
!   pdsyevx
!   rlocal_to_rglobal
!
      use parameters
      use scale
      use diffusion
      use fourier
      implicit none

      integer :: n, ix, jx, iy, jy, k, i, j, stat, it, ib, ig
      integer :: num_errors, lwork, ilwork
      integer, allocatable :: iwork(:)
      real(long), allocatable :: work(:)
!
!   a contains the sine expansion coefficients of the initial
!       temperature profile.
!   b contains the eigenvectors of the diffusion operator in the
!       sine function basis.
!   ab contains the initial temperature profile expanded in the

```



```

!           eigenvectors of the diffusion operator.
!   f contains the matrix elements of the diffusion operation in
!           sine function basis.
!   lambda contains the eigenvalues of the diffusion operator.
!
!   df contains the Fourier transform of the diffusion coefficient function.
!
real(long), allocatable :: lambda(:), xg(:, :)
real(long), allocatable :: gap(:)
real(long) :: dum

real(long), pointer :: f(:, :), b(:, :), a(:, :)
type (g_index), pointer :: f_i, b_i, a_i
integer :: f_d(DESC_DIM), b_d(DESC_DIM), a_d(DESC_DIM)

real(long), pointer :: x(:, :), ab(:, :), abt(:, :)
type (g_index), pointer :: x_i, ab_i, abt_i
integer :: x_d(DESC_DIM), ab_d(DESC_DIM), abt_d(DESC_DIM)

real(long), allocatable :: xsines(:, :), ysines(:, :), temp(:, :)
integer, allocatable :: ifail(:), iclustr(:)
integer :: biga_index, num_eigvalues, num_vectors, info

!
!   Read in the problem size, initialize the problem dimensions,
!   choose functional form for the spatially dependent heat diffusion
!   coefficients, choose functional form of initial temperature distribution
!   and choose the number of points, in both space and time, of the solution
!   to print out.
!
call init_diffusion
num_errors = 0

!
!   Read in how many sine functions to include in both the
!   x and y directions.
!
!   Because we need to get the Fourier coefficients of the sums
!   and differences of the indices, we need to include twice as
!   many Fourier coefficients as the number of sine expansion coefficients.
!   Also, because the top and bottom halves of the Fourier
!   transform are identical,
!   an artifact of artificially extending the diffusion coefficient
!   function and the initial temperature distribution, we need to
!   double the number of Fourier coefficients again. Finally, the
!   extra sum of one comes from the fact that the sine function
!   expansion starts a 1 and the Fourier expansion starts at 0.
!
!   n is the order of the diffusion operator equation.
!   n = dif_nx * dif_ny
!
!   Initialize BLACS and calculate default block sizes.
!
call initialize_scale(n, biga_index)

!
!
!   Allocate room for the eigenvalues of diffusion operator.
!
allocate( lambda(n), stat=stat)
if( stat .ne. 0) num_errors = num_errors + 1

!
!   Allocate room for sines of x and y coordinates.
!

```



```

        call get_diffusion_matrix(f,f_i,f_d)
!
!   This last call determines the matrix elements defined by equation 10
!   in the discussion paper.
!

!
!   Here we precalculate the sine functions, sin(kx) and sin(jy) used
!   in equation 13 of the discussion paper.
!   These sine functions are only evaluated at the points x and y for
!   which the solution is evaluated.
!

        do i = 1, dif_nx
            do j = 1, dif_npts
                xsines(j,i) = sqrt(2.d0/pi) * sin( i * dif_x(j))
            enddo
        enddo

        do i = 1, dif_ny
            do j = 1, dif_npts
                ysines(j,i) = sqrt(2.d0/pi) * sin( i * dif_y(j))
            enddo
        enddo

!
!   Allocate arrays for eigenvalue decomposition.
!

        allocate( ifail(n), stat=stat)
        if( stat .ne. 0) num_errors = num_errors + 1
        allocate( iclustr(n), stat=stat)
        if( stat .ne. 0) num_errors = num_errors + 1
        allocate( gap(sc_nnodes), stat=stat)
        if( stat .ne. 0) num_errors = num_errors + 1

!
!   Allocate scratch space for the symmetric eigenvector solver.
!

        lwork = 20*n + max( 5*n, n*(n+ sc_nnodes-1)/sc_nnodes )
&          + n*( (n+ sc_nnodes-1)/sc_nnodes) + 2*f_d(MB_) * f_d(MB_)

        ilwork = 2*n + max((3*n+1+sc_nnodes),max(4*n,14))

        allocate( work(lwork) , stat=stat )
        if( stat .ne. 0) num_errors = num_errors + 1
        allocate( iwork(ilwork) , stat=stat )
        if( stat .ne. 0) num_errors = num_errors + 1

!
!   Test to see if we had any allocation errors.
!

        call igamx2d(sc_icontext,'A',' ',1,1,num_errors,1,-1,-1,-1,
&          -1,-1)
&
        if( num_errors .gt. 0 ) then
            if( sc_iam .eq. 0 ) then
                write(*,*) 'Error in allocating arrays for pdsyevx in ',
&          'main'
&
                call blacs_abort(sc_icontext, 1)
            endif
        endif

        do i = 1, sc_nnodes
            gap(i) = 0.d0
        enddo

```

```

do i = 1, n
    ifail(i) = 0
    iclustr(i) = 0
enddo
!
! The call to pdsyevx will find both the eigenvalues and eigenvectors
! of the diffusion matrix operator f. The eigenvalues will be stored in
! the vector lambda and the corresponding eigenvectors will be stored in
! the matrix b. The f and b matrices in the program correspond to the
! F and B matrices in equations 11 and 12 in the
! discussion paper.
!
!
call pdsyevx('V','A','U',n,f,1,1,f_d,-1.d30,1.d30,0,n,      &
& 0.d0,num_eigvalues,num_vectors,lambda,1.d-5,b,1,1,b_d,    &
& work,lwork,iwork,ilwork,ifail,iclustr,gap,info)

if( sc_iam .eq. 0) then
    if( info .ne. 0 ) then
        write(*,*) ' info is ', info
        call blacs_abort(scicontext, 1)
    endif
endif
!
! Multiply the transpose of the eigenvector matrix, b, with the sine
! expansion of the initial temperature profile, a, to obtain the
! initial temperature profile in terms of the eigenfuctions of the
! diffusion operator.
!
call pdgemv('T', n, n, 1.d0, b, 1, 1, b_d, a, 1, 1, a_d, 1,      &
& 0.d0, ab, 1, 1, ab_d, 1)
!
! This first matrix multiplication, yielding the matrix ab,
! corresponds to the inner summation in equation 10
! of the discussion paper.
!
!
! Calculate temperature profile for each time step.
!
do it = 1, dif_ntemps
    i = 0
    do ib = 1, ab_i%num_col_blks
        do ig = ab_i%scb(ib), ab_i%ecb(ib)
            i = i + 1
            abt(1,i) = exp( -lambda(ig) * it * dif_delta_t) * ab(1,i)
        enddo
    enddo
!
! abt now has the expansion of the temperature profile in terms of the
! eigenvectors of the diffusion operator.
!
!
! Multiply the eigenvector matrix with abt to give the sine function
! expansion of the temperature profile at time t, x.
!
call pdgemv('N', n, n, 1.d0, b, 1, 1, b_d, abt, 1, 1, abt_d,      &
& 1, 0.d0, x, 1, 1, x_d, 1)
! This last sum corresponds to the outer sum of equation 12, where the
! time dependent expansion coefficients  $a_{\{sub\}1}(t)$  are stored in the
! temporary array x in the program.
!
!
! Gather all of the local pieces of the array x to the array xg.
!

```

```

call rlocal_to_rglobal(x, x_d, xg )

do k = 1, dif_npts
  temp(k, it) = 0.d0
enddo

do iy = 1, dif_ny
  do ix = 1, dif_nx
    i = (iy -1) * dif_nx + ix
    do k = 1, dif_npts
      temp(k,it) = temp(k,it) + xsines(k,ix) * ysines(k,iy)
    &
      * xg(1,i)
    enddo
  enddo
enddo

!
! This last do loop corresponds to the double summation in equation
! 13 of the discussion paper.
!

enddo ! end of time loop

!
! Here, we are just writing out the temperatures at the selected times
! and points.
!

if( sc_iam .eq. 0 ) then ! if I am node 0
  write(*,*) ' point #      X      Y'
  do i = 1, dif_npts
    write(*,'(2x, i6, 2x, 2f11.4)') i, dif_x(i), dif_y(i)
  enddo
  write(*,*)
  do k = 1, dif_npts, 6
    write(*,*)
    write(*,'(30X,'Points'))
    write(*,'('' TIME '' ,6(5x,'#'' , i4))') (i, i=k, k+5)
    do i = 1, dif_npts
      write(*,'(7f10.5)') i*dif_delta_t,
    &
      (temp(j,i),j=k,min(k+5,dif_npts))
    enddo
  enddo
endif

deallocate(xg)
deallocate(xsines)
deallocate(ysines)
deallocate(lambda)
deallocate(temp)
deallocate( ifail)
deallocate( iclustr)
deallocate( gap)
stop
end

```

## Module Parameters

```

module parameters

!
! Purpose: Define system wide parameters and index structure
!         used to help map global indices to local indices.
!

implicit none
public
integer, parameter :: long=8, short=4
real(long), parameter :: pi = 3.141592653589793d0
real(long), parameter :: twopi = 2.d0*pi

```

```

        type g_index
            integer :: num_row_blks, num_col_blks
            integer, pointer :: srb(:), scb(:), erb(:), ecb(:)
        end type g_index
!
! The g_index type was created for convenience
! components:
!     num_row_blks: number of block repetitions over matrix rows.
!     num_col_blks: number of block repetitions over matrix columns.
!     srb: global row index at start of a block
!           corresponding local index is ( block # -1) * mb
!           where mb is the number of rows in the block.
!
!     scb: global column index at start of a block
!           corresponding local index is ( block # -1) * nb
!           where nb is the number of columns in the block.
!
!     erb: last global row index in the block.
!     ecb: last global column index in the block.
!
        public g_index

    end module parameters

```

## Module Diffusion

```

    module diffusion
!
! Purpose: Assign problem parameters and initial data.
!
! Routines called:
!     none
!
!     use parameters
!     use scale
!     implicit none
!     private
!
! Make all entities private by default.
! Have all public entities have the prefix dif_.
!
! The following are the publicly available routines.
!
        public init_diffusion, init_temp, diff_coef
!
! The following are publicly available variables.
!
        real, public :: dif_ly_ratio
        integer, public :: dif_nx, dif_ny, dif_npts, dif_ntemps
        real(long), public :: dif_delta_t
        real(long), public, allocatable :: dif_x(:), dif_y(:)
!
! dif_ly_ratio is the ratio of the x and y lengths of the beam.
! dif_nx is the number of sine expansion coefficients to use
!           in the x direction.
! dif_ny is the number of sine expansion coefficients to use
!           in the y direction.
! dif_delta_t is the size of the time step to be display on output.
! dif_ntemps is the total number of temperatures to display per point.
! dif_npts is the total number of points to output.
! dif_x is the x coordinates of the points.
! dif_y is the y coordinates of the points.
!
!

```

```

!
!   Private variables
!
      integer :: init_f=1, diff_f=1
!   init_f chooses the functional form of initial distribution of temperature.
!   diff_f chooses the functional form for spatially dependent head diffusion
!       coefficient.

      contains

!*****!
!*                                     *!
!*   Module routine init_diffusion                                     *!
!*                                     *!
!*   Purpose: Initialize problem size, number of output point and *!
!*             functional form of diffusion constant and initial temperature *!
!*             distribution *!
!*                                     *!
!******!
      subroutine init_diffusion
      namelist /input/ ly_ratio, delta_t, numx, numy, nx, ny, numt,      &
&             init_f, diff_f
      integer :: numx=5, numy=5, nx=7, ny=7, numt=20
      real(long) :: ly_ratio=1.d0, delta_t=0.1
      real(long) :: delx, dely
      integer :: i, j, ij
      logical :: ex
!=====!
!           Start of executable code           !
!
      inquire ( file='diffus.nam1', exist=ex)
      if( ex ) then
        open( 10, file='diffus.nam1', action='read')
        read( 10, input)
        close(10)
      endif

      dif_ly_ratio = ly_ratio
      dif_npts = numx*numy
      dif_delta_t = delta_t
      dif_ntemps = numt
      dif_nx = nx
      dif_ny = ny
      allocate( dif_x(numx*numy) )
      allocate( dif_y(numy*numx) )

!
! Assign a simple linear array of points.
!
      delx = PI/ ( numx + 1.d0)
      dely = PI/ ( numy + 1.d0)
      do i = 1, numx
        do j = 1, numy
          ij = numx*(j-1) + i
          dif_x(ij) = delx* i
          dif_y(ij) = dely * j
        enddo
      enddo
      return
      end subroutine init_diffusion

!*****!
!*                                     *!
!*   Module routine init_temp                                     *!
!*                                     *!
!*   Purpose: Return the initial temperature of the bar at a particular *!
!*             point *!
!*                                     *!
!*****!

```

```

!*                                                                    *!
!*****!
      function init_temp(x, y)
!
!   Arguments:
!       x: real*8 (in), x coordinate
!       y: real*8 (in), y coordinate
!   Function return:
!       init_temp: real*8 (out), initial temperature at (x,y)
!
      real(long), intent(in) :: x, y
      real(long) :: init_temp

!
!   The problem has been scaled to go from 0 to pi in both the x
!   and y directions. To calculate the expansion coefficients, we
!   define the function to be odd about pi and use the range 0 < x < 2*pi
!
!   Local variables.
      integer :: isign
      real(long) :: x1, y1
!
      isign = 1
      x1 = x
      if( x .gt. pi ) then
          isign = -isign
          x1 = twopi - x
      endif
      y1 = y
      if( y .gt. pi ) then
          isign = -isign
          y1 = twopi - y
      endif

!
!   Choose very simple temperature profile cases.
!
      select case (init_f)
      case (1)
          init_temp = isign*(x1*(pi-x1))*y1*(pi-y1)
      case (2)
          init_temp = isign*(x1*(pi-x1))*y1*(pi-y1)*y1
      case (3)
          init_temp = isign*(x1*(pi-x1))*y1*(pi-y1)*x1
      case (4)
          init_temp = isign*(x1*(pi-x1))*y1*(pi-y1)*x1*y1
      case (5)
          init_temp = isign*(x1*(pi-x1))*y1*(pi-y1)
      case (6)
          init_temp = isign*(x1*(pi-x1))**2 *y1*(pi-y1)
      case (7)
          init_temp = isign*(x1*(pi-x1))*(y1*(pi-y1))**2
      case default
          init_temp = isign*sin(x1)*sin(y1)
      end select
      return
      end function init_temp

!*****!
!*                                                                    *!
!*   Module routine diff_coef                                         *!
!*                                                                    *!
!*   Purpose: Return the value of the thermal diffusion coefficient at *!
!*           an arbitrary point                                       *!
!*                                                                    *!

```



```

!*****!
      function diff_coef(x, y)
!   Arguments:
!     x: real*8 (in), x coordinate
!     y: real*8 (in), y coordinate
!   Function return:
!     diff_coef: real*8 (out), diffusion coefficient at (x,y)
!
!     real(long), intent(in) :: x, y
!     real(long) :: diff_coef
!
!
!   The problem has been scaled to go from 0 to pi in both the x
!   and y directions. To simplify the matrix element calculations,
!   we define the function to be even about pi.
!
!
!   Local variables.
!     real(long) :: x1, y1
!
!=====!
!   Start of executable code.
!
!     x1 = x
!     if( x .gt. pi ) x1 = twopi - x
!     y1 = y
!     if( y .gt. pi ) y1 = twopi - y
!
!
!   Choose very simple diffusion coefficient cases.
!
!     select case (diff_f)
!       case (1)
!         diff_coef = .5d0 + (x1 + y1) / (2 * twopi)
!       case (2)
!         diff_coef = ((1.d0 + x1)*(pi - x1 + 1.d0)*(y1 + pi))/ 3*pi
!       case (3)
!         diff_coef = (y1 + pi) * pi/((pi + x1) * (2* pi - x1))
!       case default
!         diff_coef = 1.d0
!       end select
!     return
!   end function diff_coef
!
!
!   end module diffusion

```

## Module Fourier

```

      module fourier
!
!   Purpose: To represent both the diffusion operator and
!            the temperature profile in a sine function basis.
!
!   Routines called:
!     blacs_abort
!     clocal_to_rglobal
!     dcft2
!     igamx2d
!     initialize_carray
!     initialize_scale
!     pdcft2
!
!   use parameters
!   use scale
!   use diffusion
!   implicit none
!   private

```

```

!
!   all entities private by default
!
!   external pdcft2
!
!   publicly available routines
!
!       public expand_temp_profile, get_diffusion_matrix
!       public g_index
!
!   publicly available variables
!
!
!   private variables
!
!       integer :: pn_fac(5) = 5*0 ! prime factors of number of nodes
!       nnodes = 2**pn_fac(1) * 3**pn_fac(2) * 5**pn_fac(3) *
!               7**pn_fac(4) * 11**pn_fac(5)
!
!   private routines
!
!       private minpower2, factor_nodes
!       contains
!
!*****!
!*                                     *!
!*   Module routine get_diffusion_matrix                                     *!
!*                                     *!
!*   Purpose: To obtain the matrix representation of the diffusion        *!
!*            operator in a sine function basis                           *!
!*                                     *!
!******!
!       subroutine get_diffusion_matrix(f,f_i, f_d)
!
!   Arguments:
!       f: real*8,dimension(:,:),(out), local part of the global matrix
!           containing the diffusion operator in sine function basis.
!       f_d: integer*4, dimension(:),(in), array descriptor for f.
!       f_i: g_type, (in), g_type structure for f, see parameter.f.
!
!       real(long), intent(out) :: f(:,:)
!       integer, intent(in) :: f_d(DESC_DIM)
!       type (g_index), intent(in) :: f_i
!
!   Local variables
!
!       df contains the diffusion coefficient before the call to pdcft2.
!       df contains the Fourier transpose of diffusion coefficients after the call.
!       dfg contains the entire Fourier transpose of df on each node.
!
!       complex(long), pointer :: df(:,:)
!       type (g_index), pointer :: df_i
!       integer :: df_d(DESC_DIM)
!
!       real(long), allocatable :: dfg(:,:)
!
!       !
!       !   ixi and iyi are arrays which, given a global index,
!       !   return the x and y offsets. Recall that the large arrays
!       !   are 4 dimensional arrays collapsed into 2 dimensions,
!       !   where i = (ix-1)*dif_ny + iy.
!       !
!       integer, allocatable :: ixi(:), iyi(:)
!       real(long) :: scale_f
!       integer :: nx, ny, ix, iy, ixp, iyp, istat, nerrs
!       integer :: idxiff, iydif, num_errors

```

```

        integer :: naux1, naux2, i, j, factor1, factor2, idum
        integer :: ib, jb, il, jl
!
!   ip is a support array for pdcft2.
!
        integer :: ip(40)
        integer :: blk_index
!
!   Fourier transform of diffusion coefficient function
        nerrs=0

        call factor_nodes()
        factor1 = 3**pn_fac(2) * 5**pn_fac(3) * 7**pn_fac(4) *      &
&               11**pn_fac(5)
!
!   Here we are trying to find the smallest number which is evenly
!   divisible by the number of processes and is larger than 4*(n+1).
!
        factor2 = (4*(dif_nx+1) + factor1 -1)/factor1
        nx = minpower2( factor2,idum) * factor1

        factor2 = (4*(dif_ny+1) + factor1 -1)/factor1
        ny = minpower2( factor2,idum) * factor1

        scale_f = 1.d0/ real(nx*ny, long)

!
!   Get storage for diffusion array.
!
        call initialize_scale(ny, blk_index)
        call initialize_carray(df, df_d, df_i, nx, ny,      &
&                             blk_index)
!
!   Here, we initialize the local part of the global array df, which
!   contains the value of the diffusion coefficient function, evenly
!   evaluated between (0, 2*pi). We do a two dimensional Fourier
!   transform on the data. Because the size of this array is so small,
!   nx by ny, and ultimately we have to transfer the whole array to
!   each node, it would probably be more efficient to do the calculation
!   locally on each node.
!
!
!   Get the value of the diffusion coefficient function at
!   the necessary points.
!
!
!   This loop can be simplified considerably. Because blocks of the
!   array are column-distributed with the block size equal to the number
!   of columns divided by the number of processes, there is only a single
!   column block. Also, because the processes are distributed in a 1 x np
!   arrangement, the local row index will equal the global row index.
!   However, the loop is perfectly general for other process arrangements
!   and is correct for this particular case.
!
        jl = 0
        do jb = 1, df_i%num_col_blks ! loop over the number of column blocks
            do j = df_i%scb(jb), df_i%ecb(jb)
                ! loop over columns in block
                ! j is a global index
                jl = jl + 1 ! jl is local array column index
                il = 0
                do ib = 1, df_i%num_row_blks ! loop over the number of row blocks
                    do i = df_i%srb(ib), df_i%erb(ib)
                        ! loop over rows in block
                        ! i is a global index
                        il = il + 1 ! il is local array row index

```

```

                                df(il,jl) = diff_coef((twopi*(i-1))/nx,
&                                (twopi*(j-1))/ny)
                                &
                                enddo
                                enddo
                                enddo
                                enddo
!
! This last loop just determined the diffusion coefficient at evenly
! spaced points along the x and y coordinates.
!
!
! Do the Fourier transform.
!
do i= 1, 40
    ip(i) = 0
enddo
! Store the array in normal mode overwriting the original array.
ip(1) = 1
ip(2) = 1

!
! Because the size of the 2d Fourier transform is nx by ny, which is much
! smaller than the size of the eigenvalue problem, this could probably
! be done serially on each node more quickly.
!
call pdcft2(df, df, nx, ny, 1,scale_f, sc_icontext, ip)
!
!
! df now has the Fourier coefficients for the diffusion coefficient
! function, which correspond to the D(tilde)(sub ij) given in the
! discussion paper.
!
! Because each process will need most of the Fourier transformed diffusion
! coefficients, it is useful to broadcast all parts of this matrix
! to each process.
!
! First allocate the index arrays.
!
num_errors=0
allocate(ixi(dif_nx*dif_ny), stat=istat)
if( istat .ne. 0 ) num_errors = num_errors + 1
allocate(iyi(dif_nx*dif_ny), stat=istat)
if( istat .ne. 0 ) num_errors = num_errors + 1
! Allocate array for holding global Fourier transform.
allocate(dfg(nx,ny), stat = istat)
if( istat .ne. 0 ) num_errors = num_errors + 1

call igamx2d(sc_icontext,'A',' ',1,1,num_errors,1,-1,-1,-1,
&            -1,-1)
&
if( num_errors .gt. 0 ) then
    if( sc_iam .eq. 0 ) then
        write(*,*) 'Error in allocating scratch arrays in ',
&            'get_diffusion_matrix'
&
        call blacs_abort(sc_icontext, 1)
    endif
endif

call clocal_to_rglobal( df, df_d, dfg )

! Here df contains only local portions of the global array, while
! dfg contains the entire global array.
!
!

```

```

! Now load up the diffusion operator
!   f(ix,iy;ix',iy').
!
!   Here we transform the 4d matrix into the 2d matrix where
!       i = (iy-1)* dif_nx + ix + 1
!   and
!       j = (iy'-1)* dif_nx + ix' + 1.
!
! First calculate the index arrays.
!
      do ix = 1, dif_nx
        do iy = 1, dif_ny
          i = (iy-1)* dif_nx + ix
          ixi(i) = ix
          iyi(i) = iy
        enddo
      enddo

!
! This final loop loads the matrix elements up for F as given in
! equation 10.
!
      jl = 0
      do jb = 1, f_i%num_col_blks ! loop over the number of column blocks
        do j = f_i%scb(jb), f_i%ecb(jb)
          ! loop over columns in block
          ! j is a global index
          jl = jl + 1 ! jl is local array column index
          iyp = iyi(j)
          ixp = ixi(j)
          il = 0
          do ib = 1, f_i%num_row_blks ! loop over the number of row blocks
            do i = f_i%srp(ib), f_i%erb(ib)
              ! loop over rows in block
              ! i is a global index
              il = il + 1 ! il is local array row index
              iy = iyi(i)
              ix = ixi(i)
              idxdiff = iabs(ix-ixp) + 1
              iydifff = iabs(iy-iyp) + 1
              f(il,jl) = ( ( ix*ixp + iy*iyp*dif_ly_ratio ) *
& (dfg(ixdiff, iydifff) - dfg(ix+ixp+1,iy+iyp+1)) &
& + ( ix*ixp - iy*iyp*dif_ly_ratio ) * &
& (dfg(ix+ixp+1,iydifff) - dfg(ixdiff,iy+iyp+1))) &
            enddo
          enddo
        enddo
      enddo
      deallocate(dfg)
      deallocate(ixi)
      deallocate(iyi)

!
! We should add routines to free df.
!
      return
    end subroutine get_diffusion_matrix

```

```

!*****!
!*                                     *!
!* Module routine expand_temp_profile *!
!*                                     *!
!* Purpose: To obtain the expansion coefficients of the initial *!
!*           temperature profile in a sine function expansion *!
!*                                     *!
!******!

```

```

        subroutine expand_temp_profile(a,a_i,a_d)
!
! Arguments:
!   a: real*8,dimension(:,:),(out), local part of the global matrix,
!       containing the sine coefficients for initial
!       temperature distribution.
!   a_d: integer*4, dimension(:),(in), array descriptor for a.
!   a_i: g_type, (in), g_type structure for a, see parameter.f.
!
        real(long), intent(out) :: a(:,:)
        integer, intent(in) :: a_d(DESC_DIM)
        type(g_index), intent(in) :: a_i

! Local variables
        complex(long), allocatable :: atmp(:,:)
        real(long), allocatable :: aux1(:), aux2(:)
        integer :: i,j, nx, ny, istat, naux1, naux2, nerrs, jl
        integer :: idum, jb, jx, jy
        real(long) :: x, y, scale_f

!
! Calculate the minimum power of 2 to hold twice the number of
! Fourier coefficients as sine coefficients. The top half of the
! Fourier coefficients will equal minus the bottom half because
! we are forcing the temperature profile to be odd.
!
        nx = minpower2( 2*(dif_nx+1), idum)
        ny = minpower2( 2*(dif_ny+1), idum)
        scale_f = -twopi / real( nx*ny,long)

        nerrs = 0

!
! Temperature profile allocation.
        allocate(atmp(nx,ny), stat=istat )
        if( istat .ne. 0 ) nerrs = nerrs + 1

!
        naux1 = 40000 + 2.28*( nx + ny)
        naux2 = 20000 + 66*( 256 + 2*max(nx , ny))

!
! Allocate work storage.
        allocate(aux1(naux1), stat=istat)
        if( istat .ne. 0 ) nerrs = nerrs + 1
        allocate(aux2(naux2), stat=istat)
        if( istat .ne. 0 ) nerrs = nerrs + 1

!
! Check for allocation errors.
!
        call igamx2d(sc_icontext,'A',' ',1,1,nerrs,1,-1,-1,-1,-1)
        if( nerrs .gt. 0 ) then
            if( sc_iam .eq. 0 ) then
                write(*,*) 'Error in allocating scratch arrays in ',
                    & 'expand_temp_profile'
                call blacs_abort(sc_icontext, 1)
            endif
        endif

!
! Fill atmp with the initial temperatures.
!
! atmp contains the initial temperature profile T(sub 0)(x,y) as used
! in equation 5 in the discussion paper.
!
        do i = 1, nx
            do j = 1, ny

```

```

        atmp(i,j) = init_temp((twopi*(i-1))/nx, (twopi*(j-1))/ny)
    enddo
enddo

!
! Do the 2d Fourier transform of atmp.
!
! First initialize.
!
! The 2d Fourier transform can be done in parallel, however it
! is such a small part of the problem, it is probably faster to do
! it serially on each node.
!
! Note that we could have used DSINF to obtain these expansion coefficients
! as well.
!
    call dcft2(1,atmp,1,nx,atmp,1,nx,nx,ny,1,scale_f ,aux1,naux1,      &
& aux2,naux2 )
    call dcft2(0,atmp,1,nx,atmp,1,nx,nx,ny,1,scale_f ,aux1,naux1,      &
& aux2,naux2 )
!
! The calls to dcft2 calculated the dual Fourier transform as
! defined by equation 5 in the discussion paper.
!

!
! This final loop is to load only those portions of the global array
! corresponding to the local portion of that array for this process.
!
    j1 = 0
    do jb = 1, a_i%num_col_blks ! loop over all column blocks
        do j = a_i%scb(jb), a_i%ecb(jb) ! j is global index
            jx = mod(j-1, dif_nx) + 2
            jy = (j-1) / dif_nx + 2
            j1 = j1 + 1
            a(1,j1) = real(atmp(jx,jy),long)
        enddo
    enddo

    deallocate(atmp)
    deallocate(aux1)
    deallocate(aux2)
    return
end subroutine expand_temp_profile

!*****!
!*                                     *!
!* Module routine factor_nodes                                     *!
!*                                     *!
!* Purpose: To obtain the powers of prime factorization of the number *!
!*          nodes, failing if the factorization is not compatible with *!
!*          FFT supported transform lengths                         *!
!*                                     *!
!*****!
subroutine factor_nodes()
! Arguments: None
!
! Local variables
integer n1, n2, l2
!
! Determine the prime factorization of nnodes, which must
! be of the form 2**n * 3**m * 5**i * 7**j * 11**k
! where m cannot be greater than 2 and i, j, and k cannot
! be greater than 1
!
    n2 = sc_nnodes

```

```

n1 = n2/11
if( n1*11 .eq. n2) then
    pn_fac(5) = 1
    n2 = n1
endif

n1 = n2/7
if( n1*7 .eq. n2) then
    pn_fac(4) = 1
    n2 = n1
endif

n1 = n2/5
if( n1*5 .eq. n2) then
    pn_fac(3) = 1
    n2 = n1
endif

n1 = n2/3
if( n1*3 .eq. n2) then
    if ( (n1/3)*3 .eq. n1 ) then
        pn_fac(2) = 2
        n2 = n1/3
    else
        pn_fac(2) = 1
        n2 = n1
    endif
endif

n1 = minpower2(n2,12)
pn_fac(1) = 12

if( n1 .ne. n2) then
    if( sc_iam .eq. 0) then
        write(*,*) 'Invalid number of nodes, it must have the form:'
        write(*,*) '2**n * 3**m * 5**i * 7**j * 11**k, where '
        write(*,*) ' n >= 0, 0<=m<=2 and 0<= i,j,k <=1 '
        write(*,*) ' choose a power of 2 for best performance'
        call blacs_abort(sc_icontext,1)
    endif
endif
end subroutine factor_nodes

```

```

!*****!
!*                                           *!
!*  Module function minpower2                               *!
!*                                           *!
!*  Purpose: To obtain the smallest number which is a power of 2 and *!
!*           greater than or equal to the input argument         *!
!*                                           *!
!*****!
function minpower2( n, log2n )
!
!  Arguments:
!    n: integer*4, (in), input number
!    log2n: integer*4, (out), log base 2 of the function result
!  Function return:
!    minpower2: integer*4 (out), smallest number, which is a power of 2
!               and greater than or equal to n.
!
!    integer n, minpower2, log2n
!
!  Local variables.
!    integer m, i
!    m=n

```



```

        if( n < 0 ) write(*,*) 'n cannot be negative'
powerloop: do i= 1, bit_size(n)
    m = m / 2
    if( m .eq. 0 ) exit powerloop
enddo powerloop
if ( 2**(i-1) .ne. n ) then
    if( 2**i < 0) write(*,*) 'n too large'
    log2n = i
    minpower2 = 2**i
else
    log2n = i-1
    minpower2 = n
endif
return
end function minpower2

end module fourier

```

## Module Scale

```

module scale
!
! Purpose: To initialize the communications and provide a few
!         communication utility routines.
!
! Routines called:
!   blacs_abort
!   blacs_get
!   blacs_gridinfo
!   blacs_gridinit
!   blacs_pininfo
!   dgebr2d
!   dgebs2d
!   numroc
!
!   use parameters
!   implicit none
!   external numroc
!
! All variables private by default.
!
!   private
!
! Publicly accessible routines follow.
!
!   public initialize_scale
!   public initialize_rarray, initialize_carray
!   public clocal_to_rglobal, rlocal_to_rglobal
!   public g_index
!
! Public variables follow.
!
!   integer, public :: sc_icontext, sc_iam, sc_nnodes
!
! Private variables follow.
!
! MAXBLOCK is the maximum block size. Here it is set to a very
! large number to force an equal noncyclic block distribution
! for the FFTs.
!
!
!   integer, public, parameter :: MAXBLOCK=50000
!   integer, public, parameter :: MAX_SC_INDEX=10
!   integer, public, parameter :: DESC_DIM=9
!   integer, public, parameter :: DTYPE_=1

```

```

integer, public, parameter :: CXTT_=2
integer, public, parameter :: M_=3
integer, public, parameter :: N_=4
integer, public, parameter :: MB_=5
integer, public, parameter :: NB_=6
integer, public, parameter :: RSRC_=7
integer, public, parameter :: CSRC_=8
integer, public, parameter :: LLD_=9
integer :: numroc
integer :: nprow, npcol, myrow, mycol

!
! The block sizes for a given array size are indexed in the
! nblock, mblock and nsize arrays so that, for a given array size,
! the block size calculation is done only once and exactly the same
! block sizes are returned for any particular array size.
!
integer :: nblock(MAX_SC_INDEX), mblock(MAX_SC_INDEX)
integer :: nsize(MAX_SC_INDEX)

!
! The number of different array sizes is limited by the fixed
! dimension of the arrays nblock, mblock and nsize.
!
integer :: icontext, iam, nnodes
integer, save :: sc_indx=0
integer, parameter :: rsrc=0, csrc=0
logical, save :: initialized = .false.

!
! All of the manipulation of the PESSL and SP variables will be
! done in this module and held privately.
!
contains

!*****!
!*                                           *!
!*  Module routine initialize_scale          *!
!*                                           *!
!*  Purpose: Initialize blacs and calculate a block size      *!
!*                                           *!
!******!
subroutine initialize_scale (n, index)
!
! Arguments:
!   n: integer*4 (in), matrix or vector size.
!   index: integer*4 (out), index into an array of block sizes.
!         Provides a mechanism for creating similar descriptor arrays.
!
!
! This routine assigns the block size based on a given
! n and returns an index, so that multiple arrays can be created with
! compatible block sizes.
!
integer n, index
integer npc, npr, i

if ( .not. initialized ) then
call blacs_pinfo( iam, nnodes )
sc_iam = iam
sc_nnodes = nnodes
!
! Get the number of nodes.
!
!
! The Fourier transform routines require that the processors
! be laid out in a 1 by nnodes arrangement.
!

```

```

        nprow = 1
        npcol = nnodes

        call blacs_get(0,0,icontext)
        sc_icontext = icontext
        call blacs_gridinit( icontext, 'R', nprow, npcol)
        sc_icontext = icontext
        call blacs_gridinfo( icontext, npr, npc, myrow, mycol)
!
!   Check that the system is gridded as expected.
!
        if( npr .ne. nprow .or. npc .ne. npcol) then
            if( iam .eq. 0) then
                write(*,*) 'number of processor rows and columns'
                write(*,*) 'incorrect ', nprow, npr, npcol, npc
                call blacs_abort(icontext,1)
            endif
            initialized = .true.
        endif

!
!   If we have already calculated a block size based on estimated
!   array size, return the index for that block size.
!
        do i = 1, sc_indx
            if( n .eq. nsize(i)) then
                index = i
                return
            endif
        enddo

!
!   Compute a block size.
!
        sc_indx = sc_indx + 1
        if( sc_indx .GT. MAX_SC_INDEX ) then
            if( iam .eq. 0 ) then
                write(*,*) 'Used more than the maximum number of '
                write(*,*) 'indices in initialize_scale, Maximum is ',      &
&                MAX_SC_INDEX
                call blacs_abort(icontext,1)
            endif
        endif

        index = sc_indx
        nsize(index) = n

!
!   Always choose a square block with a maximum dimension of MAXBLOCK.
!
        if( ( n ) / nnodes .gt. MAXBLOCK ) then
            mblock(index) = MAXBLOCK
        else
            mblock(index) = ( n ) / nnodes
        endif
        if( mblock(index) .lt. 1 ) then
            if( iam .eq. 0 ) then
                write(*,*) 'problem size too small for number of nodes'
                write(*,*) 'try increasing the nx and ny'
                write(*,*) n, nnodes
                call blacs_abort(sc_icontext, 1)
            endif
        endif
        nblock(index) = mblock(index)

        return

```

```

end subroutine initialize_scale

!
!   This routine is provided to allocate dynamically the space
!   needed for the local part of a global array and initialize the
!   associated array descriptor. It also returns array-useful
!   indices to do local to global index conversion.
!
!   initialize_rarray => real array initialization.
!   initialize_carray => complex array initialization.
!

!*****!
!*                                     *!
!*   Module routine initialize_rarray                                     *!
!*                                     *!
!*   Purpose: Allocate space for a real array and create associated      *!
!*             descriptor array and index array                          *!
!*                                     *!
!******!
subroutine initialize_rarray( array, desc_array,                                &
&                               index_array, m, n, blk_index)
!
!   Arguments:
!   array: pointer to real*8 (out), pointer to real array, initialized.
!   desc_array: integer*4 (out), empty descriptor array, initialized.
!   index_array: g_index (out), pointer to g_index structure,
!               see parameter.f, initialized.
!   m: integer*4 (out), scalar number of rows in global array.
!   n: integer*4 (out), scalar number of columns in global array.
!   blk_index: integer*4 (in), index into array of block sizes to use
!               for initializing the descriptor array.
!
!   integer, intent(out) :: desc_array(DESC_DIM)
!   integer, intent(in) :: m, n, blk_index
!   type (g_index), pointer :: index_array
!   real(long), pointer :: array(:, :)

! Local variables
integer :: irows, icols, istat, i, j
integer :: start_row_block, start_col_block
integer :: mb, nb, num_mb, num_nb

!
!   Check to see if the block sizes were already calculated.
!
!   if ( blk_index .lt. 1 .or. blk_index .gt. sc_indx ) then
!       if( iam .eq. 0 ) then
!           write(*,*) 'No initialization done for index ', blk_index
!           call blacs_abort(icontext, 1)
!       endif
!   endif

mb = mblock(blk_index)
nb = nblock(blk_index)

irows = numroc( m, mb, myrow, rsrc, nprow )
icols = numroc( n, nb, mycol, csrc, npc )

allocate(array(max(irows,1),max(icols,1)), stat=istat)
if ( istat .ne. 0 ) then
    write(*,*) 'allocate failed in initialize_array'
    call blacs_abort(icontext,1)
endif

!

```

```

! Calculate the number of row and column blocks.
!
      num_mb = ( (irows + mb -1)/ mb )
      num_nb = ( (icols + nb -1)/ nb )

      allocate (index_array)
      index_array%num_row_blks = num_mb
      index_array%num_col_blks = num_nb

      allocate(index_array%srb(num_mb))
      allocate(index_array%erb(num_mb))
      allocate(index_array%scb(num_nb))
      allocate(index_array%ecb(num_nb))

      desc_array(DTYPE_) = 1
      desc_array(M_) = m
      desc_array(N_) = n
      desc_array(MB_) = mb
      desc_array(NB_) = nb
      desc_array(RSRC_) = rsrc
      desc_array(CSRC_) = csrc
      desc_array(CTXT_) = icontext
      desc_array(LLD_) = max(irows,1)

!
      start_row_block = mod( nprow + myrow - rsrc , nprow )
      start_col_block = mod( npcol + mycol - csrc , npcol )

      do i = 1, index_array%num_row_blks
         index_array%srb(i) = (start_row_block + (i - 1)*nprow) *      &
&           mb+1
         index_array%erb(i) = index_array%srb(i) + mb - 1
      enddo
      index_array%erb(num_mb) = mod(irows-1,mb) +      &
&           index_array%srb(num_mb)

      do i = 1, index_array%num_col_blks
         index_array%scb(i) = (start_col_block + (i - 1)*npcol) *      &
&           nb + 1
         index_array%ecb(i) = index_array%scb(i) + nb - 1
      enddo
      index_array%ecb(num_nb) = mod(icols-1,nb) +      &
&           index_array%scb(num_nb)

      end subroutine initialize_rarray

! Complex array initialization.
!
!*****!
!*                                           *!
!*  Module routine initialize_carray          *!
!*                                           *!
!*  Purpose: Allocate space for a complex array and create associated      *!
!*           descriptor array and index array                                *!
!*                                           *!
!*****!
      subroutine initialize_carray( array, desc_array,      &
&           index_array, m, n, blk_index)
!
! Arguments:
!   array: pointer to complex (out), pointer to real array, initialized.
!   desc_array: integer*4 (out), empty descriptor array, initialized.
!   index_array: g_index (out), pointer to g_index structure,
!               see parameter.f, initialized.
!   m: integer*4 (out), scalar number of rows in global array.
!   n: integer*4 (out), scalar number of columns in global array.

```

```

!      blk_index: integer*4 (in), index into array of block sizes to use
!                  for initializing the descriptor array.
!
      integer desc_array(DESC_DIM), m, n, blk_index
      type (g_index), pointer :: index_array
      complex(long), pointer :: array(:, :)

! Local variables
      integer :: irows, icols, istat, i, j
      integer :: start_row_block, start_col_block
      integer :: mb, nb, num_mb, num_nb

!
!      Check to see if the block sizes were already calculated.
!
      if ( blk_index .lt. 1 .or. blk_index .gt. sc_indx ) then
        if( iam .eq. 0 ) then
          write(*,*) 'No initialization done for index ', blk_index
          call blacs_abort(icontext, 1)
        endif
      endif

      mb = mblock(blk_index)
      nb = nblock(blk_index)

      irows = numroc( m, mb, myrow, rsrc, nprow )
      icols = numroc( n, nb, mycol, csrc, npcyl )

      allocate(array(max(irows,1),max(icols,1)), stat=istat)
      if ( istat .ne. 0 ) then
        write(*,*) 'allocate failed in initialize_array'
        call blacs_abort(icontext,1)
      endif

!
!      Calculate the number of row and column blocks.
!
      num_mb = ( (irows + mb -1)/ mb )
      num_nb = ( (icols + nb -1)/ nb )

      allocate(index_array, stat=istat)
      if ( istat .ne. 0 ) then
        write(*,*) 'allocate failed in initialize_array'
        call blacs_abort(icontext,1)
      endif

      index_array%num_row_blks = num_mb
      index_array%num_col_blks = num_nb

      allocate(index_array%srbs(num_mb))
      allocate(index_array%erbs(num_mb))
      allocate(index_array%scbs(num_nb))
      allocate(index_array%ecbs(num_nb))

      desc_array(DTYPE_) = 1
      desc_array(M_) = m
      desc_array(N_) = n
      desc_array(MB_) = mb
      desc_array(NB_) = nb
      desc_array(RSRC_) = rsrc
      desc_array(CSRC_) = csrc
      desc_array(CTXT_) = icontext
      desc_array(LLD_) = max(irows,1)

```

```

!
      start_row_block = mod( nprow + myrow - rsrc , nprow )
      start_col_block = mod( npcol + mycol - csrc , npcol )

      do i = 1, index_array%num_row_blks
        index_array%srb(i) = (start_row_block + (i - 1)*nprow) *
&                               mb+1 &
        index_array%erb(i) = index_array%srb(i) + mb - 1
      enddo
      index_array%erb(num_mb) = mod(irows-1,mb) +
&                               index_array%srb(num_mb) &

      do i = 1, index_array%num_col_blks
        index_array%scb(i) = (start_col_block + (i - 1)*npcol) *
&                               nb + 1 &
        index_array%ecb(i) = index_array%scb(i) + nb - 1
      enddo
      index_array%ecb(num_nb) = mod(icols-1,nb) +
&                               index_array%scb(num_nb) &

      end subroutine initialize_carray

!
!*****!
!*                                     *!
!*  Module routine clocal_to_rglobal                                     *!
!*                                     *!
!*  Purpose: Take the real parts of the local portions of a complex matrix *!
!*            and distribute them globally to each node                                     *!
!*                                     *!
!*****!
      subroutine clocal_to_rglobal(a,a_d,a_glob)
!
!  Arguments:
!    a: complex*16, dimension(:,.), is the local part of a
!                                     global complex array.
!    a_d: integer*4, array descriptor for a.
!    a_glob: real*8, dimension(:,.), entire matrix A on each node.
!
      complex(long), intent(in) :: a(:,.)
      integer, intent(in) :: a_d(DESC_DIM)
      real(long), intent(out) :: a_glob(:,.)
!
!  Local variables.
!
      integer :: nrow_blks, ncol_blks, ib, jb, ibl, jbl, i, j
      integer :: m, n, nb, mb, ig, jg, lda, il, jl, prow, pcol
      integer :: iarow, iacol, ni, nj
!
!  m is number of rows in global matrix.
!  n is number of columns in global matrix.
!  mb and nb are rows and columns in each block.
!  prow and pcol are the processor row and column containing a block.
!  ib, jb, ibl, jbl are global and local block indices.
!  il, jl, ig, jg are local and global matrix indices.
!
!
!
!  Start of executable code.
!
!=====

```

```

!
! Determine the total number of row and column blocks.
      m = a_d(M_)
      n = a_d(N_)
      mb = a_d(MB_)
      nb = a_d(NB_)
      iarow = a_d(RSRC_)
      iacol = a_d(CSRC_)
      nrow_blks = ( m + mb - 1 ) / mb
      ncol_blks = ( n + nb - 1 ) / nb
!
! Determine leading dimension of global array.
      lda = size(a_glob, dim=1)

!
! Loop over all of the blocks.
!
      do jb = 0, ncol_blks - 1
!
! Loop over column blocks, determining both local and global j indices.
      jg = jb * nb + 1
      nj = nb
      if (jb .eq. ncol_blks - 1) nj = mod( n - 1, nb ) + 1
      jbl = ( jb + iacol ) / npcol - (mycol + iacol) / npcol
      jl = jbl * nb + 1
      pcol = mod((jb + iacol), npcol )

      do ib = 0, nrow_blks - 1
!
! Loop over row blocks, determining both local and global i indices.
      ig = ib * mb + 1
      ni = mb
      if (ib .eq. nrow_blks - 1) ni = mod( m - 1, mb ) + 1
      ibl = ( ib + iarow ) / nprow - (myrow + iarow) / nprow
      il = ibl * mb + 1
      prow = mod((ib + iarow), nprow )

!
! Determine if this block is on my processor.
!
      if( prow .eq. myrow .and. pcol .eq. mycol) then
!
! Block is on my processor.
!
! using Fortran 90 array language this is
!      a_glob(ig:ig+ni-1,jg:jg+nj-1) = a(il:il+ni-1:jl+jn-1)
!
!
!      do j = 1, nj
!        do i = 1, ni
!          a_glob( ig+i-1, jg+j-1 ) = a( il+i-1, jl+j-1)
!        enddo
!      enddo
!      call dgebs2d(icontext,'ALL',' ',ni,nj,a_glob(ig,jg),lda)
!    else
!
! Block is on somebody elses processor.
!
!      call dgebr2d(icontext,'ALL',' ',ni,nj,a_glob(ig,jg),
!
!      &      lda, prow, pcol)
!
!      endif
!
!    enddo
!  enddo

  return
end subroutine clocal_to_rglobal

```



```

!*****!
!*                                           *!
!*  Module routine rlocal_to_rglobal          *!
!*                                           *!
!*  Purpose: Take the local portions of a real matrix      *!
!*           and distribute them globally to each node     *!
!*                                           *!
!*****!
      subroutine rlocal_to_rglobal(a,a_d,a_glob)
!
!  Arguments:
!    a: real*8, dimension(:,:), is the local part of a global real array.
!    a_d: integer*4, array descriptor for a.
!    a_glob: real*8, dimension(:,:), entire matrix A on each node.
!
!  Input arguments.
!
!      real(long), intent(in) :: a(:, :)
!      integer, intent(in) :: a_d(DESC_DIM)
!      real(long), intent(out) :: a_glob(:, :)
!
!  Local variables.
!
!      integer :: nrow_blks, ncol_blks, ib, jb, ibl, jbl, i, j
!      integer :: m, n, nb, mb, ig, jg, lda, il, jl, prow, pcol
!      integer :: iarow, iacol, ni, nj
!
!  m is number of rows in global matrix.
!  n is number of columns in global matrix.
!  mb and nb are rows and columns in each block.
!  prow and pcol are the processor row and column containing a block.
!  ib, jb, ibl, jbl are global and local block indices.
!  il, jl, ig, jg are local and global matrix indices.
!
!
!
!  Start of executable code.
!
!=====
!
! Determine the total number of row and column blocks.
!      m = a_d(M_)
!      n = a_d(N_)
!      mb = a_d(MB_)
!      nb = a_d(NB_)
!      iarow = a_d(RSRC_)
!      iacol = a_d(CSRC_)
!      nrow_blks = ( m + mb - 1 ) / mb
!      ncol_blks = ( n + nb - 1 ) / nb
!
! Determine leading dimension of global array.
!      lda = size(a_glob, dim=1)
!
! Loop over all of the blocks.
!
!      do jb = 0, ncol_blks - 1
!
!  Loop over column blocks, determining both local and global j indices
!      jg = jb * nb + 1
!      nj = nb
!      if (jb .eq. ncol_blks - 1) nj = mod( n - 1, nb ) + 1
!      jbl = ( jb + iacol ) / npcol - (mycol + iacol) / npcol

```

```

        j1 = jbl * nb + 1
        pcol = mod((jb + iacol), npcol )

        do ib = 0, nrow_blks - 1
!
!   Loop over row blocks, determining both local and global i indices.
        ig = ib * mb + 1
        ni = mb
        if (ib .eq. nrow_blks - 1) ni = mod( m - 1, mb) + 1
        ibl = ( ib + iarow ) / nprow - (myrow + iarow) / nprow
        il = ibl * mb + 1
        prow = mod((ib + iarow), nprow )
!
!   Determine if this block is on my processor.
!
        if( prow .eq. myrow .and. pcol .eq. mycol) then
!
!   Block is on my processor.
!
            do j = 1, nj
                do i = 1, ni
                    a_glob( ig+i-1, jg+j-1 ) = a( il+i-1, j1+j-1)
                enddo
            enddo
            call dgebs2d(icontext,'ALL',' ',ni,nj,a_glob(ig,jg),lda)
        else
!
!   Block is on somebody elses processor.
!
            call dgebr2d(icontext,'ALL',' ',ni,nj,a_glob(ig,jg),lda, prow, pcol)
        &
        endif

        enddo
    enddo

    return
end subroutine rlocal_to_rglobal

end module scale

```

## Input Data

```

&input
ly_ratio = 1.0, delta_t = .1, nx =15, ny=15, numx = 5, numy =5,
numt=20, init_f=1, diff_f =3
/

```

## Output Data

0:	point #	X	Y
0:	1	0.5236	0.5236
0:	2	1.0472	0.5236
0:	3	1.5708	0.5236
0:	4	2.0944	0.5236
0:	5	2.6180	0.5236
0:	6	0.5236	1.0472
0:	7	1.0472	1.0472
0:	8	1.5708	1.0472
0:	9	2.0944	1.0472
0:	10	2.6180	1.0472
0:	11	0.5236	1.5708
0:	12	1.0472	1.5708
0:	13	1.5708	1.5708
0:	14	2.0944	1.5708
0:	15	2.6180	1.5708
0:	16	0.5236	2.0944



0:	0.90000	1.89793	1.62712	0.91312	0.70885	1.26279	1.47284
0:	1.00000	1.65761	1.42087	0.79712	0.61636	1.09824	1.28104
0:	1.10000	1.44792	1.24100	0.69609	0.53650	0.95604	1.11524
0:	1.20000	1.26492	1.08410	0.60802	0.46739	0.83291	0.97163
0:	1.30000	1.10520	0.94718	0.53119	0.40745	0.72611	0.84705
0:	1.40000	0.96575	0.82766	0.46415	0.35539	0.63334	0.73883
0:	1.50000	0.84399	0.72330	0.40561	0.31012	0.55266	0.64471
0:	1.60000	0.73764	0.63215	0.35450	0.27071	0.48243	0.56279
0:	1.70000	0.64474	0.55254	0.30985	0.23638	0.42125	0.49142
0:	1.80000	0.56358	0.48298	0.27084	0.20646	0.36792	0.42920
0:	1.90000	0.49265	0.42220	0.23676	0.18036	0.32140	0.37493
0:	2.00000	0.43067	0.36908	0.20697	0.15758	0.28081	0.32758
0:							
0:				Points			
0:	TIME	# 19	# 20	# 21	# 22	# 23	# 24
0:	0.10000	4.13480	2.45333	1.42982	2.41948	2.75758	2.41948
0:	0.20000	3.53366	2.04155	1.14813	1.99356	2.29549	1.99356
0:	0.30000	3.02543	1.72459	0.95015	1.67033	1.93488	1.67033
0:	0.40000	2.59910	1.47101	0.79960	1.41458	1.64393	1.41458
0:	0.50000	2.23988	1.26277	0.67989	1.20679	1.40486	1.20679
0:	0.60000	1.93538	1.08877	0.58206	1.03495	1.20593	1.03495
0:	0.70000	1.67587	0.94166	0.50068	0.89108	1.03879	0.89108
0:	0.80000	1.45370	0.81630	0.43217	0.76952	0.89732	0.76952
0:	0.90000	1.26279	0.70885	0.37401	0.66612	0.77685	0.66612
0:	1.00000	1.09824	0.61636	0.32432	0.57769	0.67376	0.57769
0:	1.10000	0.95604	0.53650	0.28168	0.50176	0.58522	0.50176
0:	1.20000	0.83291	0.46739	0.24495	0.43633	0.50892	0.43633
0:	1.30000	0.72611	0.40745	0.21322	0.37981	0.44300	0.37981
0:	1.40000	0.63334	0.35539	0.18576	0.33088	0.38592	0.33088
0:	1.50000	0.55266	0.31012	0.16193	0.28844	0.33642	0.28844
0:	1.60000	0.48243	0.27071	0.14124	0.25158	0.29343	0.25158
0:	1.70000	0.42125	0.23638	0.12325	0.21953	0.25604	0.21953
0:	1.80000	0.36792	0.20646	0.10759	0.19163	0.22350	0.19163
0:	1.90000	0.32140	0.18036	0.09395	0.16733	0.19516	0.16733
0:	2.00000	0.28081	0.15758	0.08205	0.14614	0.17045	0.14614
0:							
0:				Points			
0:	TIME	# 25	# 26	# 27	# 28	# 29	# 30
0:	0.10000	1.42982					
0:	0.20000	1.14813					
0:	0.30000	0.95015					
0:	0.40000	0.79960					
0:	0.50000	0.67989					
0:	0.60000	0.58206					
0:	0.70000	0.50068					
0:	0.80000	0.43217					
0:	0.90000	0.37401					
0:	1.00000	0.32432					
0:	1.10000	0.28168					
0:	1.20000	0.24495					
0:	1.30000	0.21322					
0:	1.40000	0.18576					
0:	1.50000	0.16193					
0:	1.60000	0.14124					
0:	1.70000	0.12325					
0:	1.80000	0.10759					
0:	1.90000	0.09395					
0:	2.00000	0.08205					

---

## Sample Sparse Linear Algebraic Equations Programs

This section contains the sample programs and utilities that use the Fortran 90 and Fortran 77 sparse linear algebraic equations subroutines. You can use the makefile shown in “Sample Makefiles and Run Script for AIX” on page 967 to build these sample programs. A copy of these programs and the makefile are provided with the Parallel ESSL product. For file names and installation procedures, see the *Parallel ESSL Installation Memo*.

The following list describes these sample programs and their utilities:

- The Fortran 90 and Fortran 77 sample programs shown in “Fortran 90 Sample Sparse Program” on page 932 and “Fortran 77 Sample Sparse Program” on page 941, respectively, solve a sparse linear system based on the discretization of the same elliptic partial differential equation:

$$-b_1 \frac{\partial^2(u)}{\partial x^2} - b_2 \frac{\partial^2(u)}{\partial y^2} - b_3 \frac{\partial^2(u)}{\partial z^2} - a_1 \frac{\partial(u)}{\partial x} - a_2 \frac{\partial(u)}{\partial y} - a_3 \frac{\partial(u)}{\partial z} + a_4 u = 0$$

with the Dirichlet boundary conditions on the unit cube, that is,  $0 \leq x, y, z \leq 1$ . The equation is discretized with finite differences and uniform stepsize, where the resulting discrete equation is:

$$u(x, y, z) (2b_1 + 2b_2 + 2b_3 + a_1 + a_2 + a_3) + u(x-1, y, z) (-b_1 - a_1) + u(x, y-1, z) (-b_2 - a_2) \\ + u(x, y, z-1) (-b_3 - a_3) - u(x+1, y, z)b_1 - u(x, y+1, z)b_2 - u(x, y, z+1) (b_3) \left( \frac{1}{h^2} \right)$$

This elliptic partial differential equation is similar to an example problem discussed in [45].

In these sample programs the index space of the discretized computational domain is first numbered sequentially in a standard way. Then, the corresponding vector is distributed using block data distribution, which is implemented using the subroutine shown in “PART\_BLOCK (Block Data Distribution)” on page 956.

Boundary conditions are set in a very simple way, by adding equations of the form:  $u(x, y, z) = rhs(x, y, z)$

From the command line, you can specify the number of gridpoints along the edges of the unit cube, the iterative solution method, the preconditioner and the stopping criterion to be used.

- The Fortran 90 sample program shown in “Fortran 90 Sample Sparse Program (using the Harwell-Boeing exchange format)” on page 950 shows how to build and solve a sparse linear system with the coefficient matrices read from external storage, using the Harwell-Boeing exchange format. Details on the Harwell-Boeing exchange format and sample matrices are available from <http://math.nist.gov/MatrixMarket/>.

From the command line, you can specify a file containing the input matrix, an iterative method and preconditioner, and a data distribution to be used.

This sample program uses the following subroutines:

- One of the following data distribution subroutines:
  - PART\_BLOCK, which implements a block data distribution
  - PARTBCYC, which implements a block-cyclic data distribution

- PARTRAND, which implements a random data distribution  
(These subroutines are documented in “Sample PARTS Subroutine” on page 956.)
- READ\_MAT, a serial module, reads a matrix in Harwell-Boeing format from a file (see “The READ\_MAT Subroutine” on page 959)
- MAT\_DIST, a utility module, scatters a sparse matrix across a process grid according to a user-specified data distribution (see “The MAT\_DIST Subroutine” on page 962)
- DESYM, a utility subroutine, takes a matrix stored in symmetric format (that is, stores only the upper or lower triangle) and converts it into full storage format, assuming storage-by-rows representation. (see “The DESYM Subroutine” on page 966)

**Note:** The performance of the iterative methods depends heavily on the choice of data distribution. The random data distribution is usually not a good choice. It is provided to serve as a template to help you implement a graph partitioning data distribution, which you can do by substituting the call to the random number generator in the PARTRAND initialization routine with a call to a graph partitioning package. The data distributions based on graph partitioning and/or physical considerations usually give the best performance; in general, experimentation is required to determine the best data distribution for your particular application.

## Fortran 90 Sample Sparse Program

```
@process free(f90) init(f90ptr) nosave
!
! This sample program shows how to build and solve a sparse linear
! system using the subroutines in the sparse section of Parallel
! ESSL. The matrix and RHS are generated
! in parallel, so that there is no serial bottleneck.
!
! The program solves a linear system based on the partial differential
! equation
!
!
!
! 
$$-\frac{b_1}{dx dx} \frac{dd(u)}{dy dy} - \frac{b_2}{dy dy} \frac{dd(u)}{dz dz} - \frac{b_3}{dz dz} \frac{dd(u)}{dx dx} - \frac{a_1}{dx} \frac{d(u)}{dy} - \frac{a_2}{dy} \frac{d(u)}{dz} - \frac{a_3}{dz} \frac{d(u)}{dx} + a_4 u$$

!
! = 0
!
! with Dirichlet boundary conditions on the unit cube
!
!  $0 \leq x, y, z \leq 1$ 
!
! The equation is discretized with finite differences and uniform stepsize;
! the resulting discrete equation is
!
! 
$$(u(x,y,z)(2b_1+2b_2+2b_3+a_1+a_2+a_3)+u(x-1,y,z)(-b_1-a_1)+u(x,y-1,z)(-b_2-a_2)+$$

! 
$$+u(x,y,z-1)(-b_3-a_3)-u(x+1,y,z)b_1-u(x,y+1,z)b_2-u(x,y,z+1)b_3)*(1/h**2)$$

!
!
! In this sample program the index space of the discretized
! computational domain is first numbered sequentially in a standard way,
! then the corresponding vector is distributed according to an HPF BLOCK
! distribution directive.
!
! Boundary conditions are set in a very simple way, by adding
! equations of the form
!
```

```

!   u(x,y,z) = rhs(x,y,z)
!
Program PDE90
USE F90SPARSE
Implicit none

INTERFACE PART_BLOCK
!   .....user defined subroutine.....
SUBROUTINE PART_BLOCK(GLOBAL_INDX,N,NP,PV,NV)
IMPLICIT NONE
INTEGER, INTENT(IN)  :: GLOBAL_INDX, N, NP
INTEGER, INTENT(OUT) :: NV
INTEGER, INTENT(OUT) :: PV(*)
END SUBROUTINE PART_BLOCK
END INTERFACE

! Input parameters
Character*10 :: CMETHD, PREC
Integer      :: IDIM, IRET

! Miscellaneous
Integer, Parameter :: IZERO=0, IONE=1
Character, Parameter :: ORDER='R'
INTEGER            :: IARGC
REAL(KIND(1.D0)), Parameter :: DZERO = 0.D0, ONE = 1.D0
REAL(KIND(1.D0)) :: TIMEF, T1, T2, TPREC, TSOLVE, T3, T4
EXTERNAL TIMEF

! Sparse Matrix and preconditioner
TYPE(D_SPMAT) :: A
TYPE(D_PRECN) :: APRC
! Descriptor
TYPE(DESC_TYPE) :: DESC_A
! Dense Matrices
REAL(KIND(1.d0)), POINTER :: B(:), X(:)

! BLACS parameters
INTEGER :: nprow, npcol, icontxt, iam, np, myprow, mypcol

! Solver parameters
INTEGER :: ITER, ITMAX, IERR, ITRACE, METHD, IPREC, ISTOPC, &
& IPARM(20)
REAL(KIND(1.D0)) :: ERR, EPS, RPARM(20)

! Other variables
INTEGER :: I, INFO
INTEGER :: INTERNAL, M, II

! Initialize BLACS
CALL BLACS_PINFO(IAM, NP)
CALL BLACS_GET(IZERO, IZERO, ICONTXT)

! Rectangular Grid, P x 1

CALL BLACS_GRIDINIT(ICONTXT, ORDER, NP, IONE)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYPROW, MYPCOL)

!
!   Get parameters
!
CALL GET_PARM(ICONTXT, CMETHD, PREC, IDIM, ISTOPC, ITMAX, ITRACE)

!
!   Allocate and fill in the coefficient matrix, RHS and initial guess
!

CALL BLACS_BARRIER(ICONTXT, 'A11')

```

```

T1 = TIMEF()
CALL CREATE_MATRIX(PART_BLOCK,IDIM,A,B,X,DESC_A,ICONXT)
T2 = TIMEF() - T1

CALL DGAMX2D(ICONXT,'A',' ',IONE, IONE,T2,IONE,T1,T1,-1,-1,-1)
IF (IAM.EQ.0) Write(6,*) 'Matrix creation Time : ',T2/1.D3

!
! Prepare the preconditioner.
!
SELECT CASE (PREC)
CASE ('ILU')
    IPREC = 2
CASE ('DIAGSC')
    IPREC = 1
CASE ('NONE')
    IPREC = 0
CASE DEFAULT
    WRITE(0,*) 'Unknown preconditioner'
    CALL BLACS_ABORT(ICONXT,-1)
END SELECT
CALL BLACS_BARRIER(ICONXT,'All')
T1 = TIMEF()
CALL PSPGPR(IPREC,A,APRC,DESC_A,INFO=IRET)
TPREC = TIMEF()-T1

CALL DGAMX2D(icontxt,'A',' ',IONE, IONE,TPREC,IONE,t1,t1,-1,-1,-1)

IF (IAM.EQ.0) WRITE(6,*) 'Preconditioner Time : ',TPREC/1.D3

IF (IRET.NE.0) THEN
    WRITE(0,*) 'Error on preconditioner',IRET
    CALL BLACS_ABORT(ICONXT,-1)
    STOP
END IF

!
! Iterative method parameters
!
IF (CMETHD(1:6).EQ.'CGSTAB') Then
    METHD = 1
ELSE IF (CMETHD(1:3).EQ.'CGS') Then
    METHD = 2
ELSE IF (CMETHD(1:5).EQ.'TFQMR') THEN
    METHD = 3
ELSE
    WRITE(0,*) 'Unknown method '
    CALL BLACS_ABORT(ICONXT,-1)
END IF
EPS = 1.D-9
IPARM = 0
RPARM = 0.D0
IPARM(1) = METHD
IPARM(2) = ISTOPC
IPARM(3) = ITMAX
IPARM(4) = ITRACE
RPARM(1) = EPS
CALL BLACS_BARRIER(ICONXT,'All')
T1 = TIMEF()
CALL PSPGIS(A,B,X,APRC,DESC_A,&
    & IPARM=IPARM,RPARM=RPARM,INFO=IERR)
CALL BLACS_BARRIER(ICONXT,'All')
T2 = TIMEF() - T1
ITER = IPARM(5)
ERR = RPARM(2)
CALL DGAMX2D(ICONXT,'A',' ',IONE, IONE,T2,IONE,T1,T1,-1,-1,-1)

```



```

IF (IAM.EQ.0) THEN
  WRITE(6,*) 'Time to Solve Matrix : ',T2/1.D3
  WRITE(6,*) 'Time per iteration : ',T2/(ITER*1.D3)
  WRITE(6,*) 'Number of iterations : ',ITER
  WRITE(6,*) 'Error on exit : ',ERR
  WRITE(6,*) 'INFO on exit : ',IERR
END IF

!
! Cleanup storage and exit
!
CALL PGEFREE(B,DESC_A)
CALL PGEFREE(X,DESC_A)

CALL PSPFREE(APRC,DESC_A)
CALL PSPFREE(A,DESC_A)

CALL PADFREE(DESC_A)

CALL BLACS_GRIDEXIT(ICONTEXT)
CALL BLACS_EXIT(0)

STOP

CONTAINS
!
! Subroutine to allocate and fill in the coefficient matrix and
! the RHS.
!
SUBROUTINE CREATE_MATRIX(PARTS,IDIM,A,B,T,DESC_A,ICONTEXT)
!
! Discretize the partial diferential equation
!
! 
$$-\frac{b_1}{dx} \frac{dd(u)}{dx} - \frac{b_2}{dy} \frac{dd(u)}{dy} - \frac{b_3}{dz} \frac{dd(u)}{dz} - \frac{a_1}{dx} \frac{d(u)}{dx} - \frac{a_2}{dy} \frac{d(u)}{dy} - \frac{a_3}{dz} \frac{d(u)}{dz} + a_4 u = 0$$

!
! = 0
!
! boundary condition: Dirichlet
! 0 < x,y,z < 1
!
! 
$$u(x,y,z)(2b_1+2b_2+2b_3+a_1+a_2+a_3)+u(x-1,y,z)(-b_1-a_1)+u(x,y-1,z)(-b_2-a_2)+$$

! 
$$+ u(x,y,z-1)(-b_3-a_3)-u(x+1,y,z)b_1-u(x,y+1,z)b_2-u(x,y,z+1)b_3$$

!
  USE F90SPARSE
  Implicit None
  INTEGER :: IDIM
  INTERFACE PARTS
    SUBROUTINE PARTS(GLOBAL_INDX,N,P,PV,NV)
      IMPLICIT NONE
      INTEGER, INTENT(IN) :: GLOBAL_INDX, N, P
      INTEGER, INTENT(OUT) :: NV
      INTEGER, INTENT(OUT) :: PV(*)
    END SUBROUTINE PARTS
  END INTERFACE
  Real(Kind(1.D0)),Pointer :: B(:),T(:)
  Type (DESC_TYPE) :: DESC_A
  Integer :: ICONTEXT
  Type(D_SPMAT) :: A
  Real(Kind(1.d0)) :: ZT(10),GLOB_X,GLOB_Y,GLOB_Z
  Integer :: M,N,NNZ,GLOB_ROW,J
  Type (D_SPMAT) :: ROW_MAT
  Integer :: X,Y,Z,COUNTER,IA,I,INDX_OWNER
  INTEGER :: NPROW,NPCOL,MYPCOL,MYPCOL
  Integer :: ELEMENT
  INTEGER :: INFO, NV, INV

```

```

INTEGER, ALLOCATABLE      :: PRV(:)
! deltax dimension of each grid cell
! deltat discretization time
Real(Kind(1.D0))          :: DELTAH
Real(Kind(1.d0)),Parameter :: RHS=0.d0,ONE=1.d0,ZERO=0.d0
Real(Kind(1.d0))          :: TIMEF, T1, T2, TINS
external                  timef
! common area

CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYPROW, MYPCOL)

DELTAH = 1.D0/(IDIM-1)

! Initialize array descriptor and sparse matrix storage. Provide an
! estimate of the number of non zeroes

M   = IDIM*IDIM*IDIM
N   = M
NNZ = (N*7)/(NPROW*NPCOL)
Call PADALL(N,PARTS,DESC_A,ICONTXT)
Call PSPALL(A,DESC_A,NNZ=NNZ)
! Define RHS from boundary conditions; also build initial guess
Call PGEALL(B,DESC_A)
Call PGEALL(T,DESC_A)

! We build an auxiliary matrix consisting of one row at a
! time
ROW_MAT%DESCRA(1:1) = 'G'
ROW_MAT%FIDA        = 'CSR'
ALLOCATE(ROW_MAT%AS(20))
ALLOCATE(ROW_MAT%IA1(20))
ALLOCATE(ROW_MAT%IA2(20))
ALLOCATE(PRV(NPROW))
ROW_MAT%IA2(1)=1

TINS = 0.D0
CALL BLACS_BARRIER(ICONTXT,'ALL')
T1 = TIMEF()

! Loop over rows belonging to current process in a BLOCK
! distribution.

DO GLOB_ROW = 1, N
  CALL PARTS(GLOB_ROW,N,NPROW,PRV,NV)
  DO INV = 1, NV
    INDX_OWNER = PRV(INV)
    IF (INDX_OWNER == MYPROW) THEN
      ! Local matrix pointer
      ELEMENT=1
      ! Compute gridpoint Coordinates
      IF (MOD(GLOB_ROW,(IDIM*IDIM)).EQ.0) THEN
        X = GLOB_ROW/(IDIM*IDIM)
      ELSE
        X = GLOB_ROW/(IDIM*IDIM)+1
      ENDIF
      IF (MOD((GLOB_ROW-(X-1)*IDIM*IDIM),IDIM).EQ.0) THEN
        Y = (GLOB_ROW-(X-1)*IDIM*IDIM)/IDIM
      ELSE
        Y = (GLOB_ROW-(X-1)*IDIM*IDIM)/IDIM+1
      ENDIF
      Z = GLOB_ROW-(X-1)*IDIM*IDIM-(Y-1)*IDIM
      ! GLOB_X, GLOB_Y, GLOB_Z coordinates
      GLOB_X=X*DELTAH
      GLOB_Y=Y*DELTAH
      GLOB_Z=Z*DELTAH
    END DO
  END DO
END DO

```

```

! Check on boundary points
IF (X.EQ.1) THEN
  ROW_MAT%AS(ELEMENT)=ONE
  ROW_MAT%IA1(ELEMENT)=(X-1)*IDIM*IDIM+(Y-1)*IDIM+(Z)
  ELEMENT=ELEMENT+1
ELSE IF (Y.EQ.1) THEN
  ROW_MAT%AS(ELEMENT)=ONE
  ROW_MAT%IA1(ELEMENT)=(X-1)*IDIM*IDIM+(Y-1)*IDIM+(Z)
  ELEMENT=ELEMENT+1
ELSE IF (Z.EQ.1) THEN
  ROW_MAT%AS(ELEMENT)=ONE
  ROW_MAT%IA1(ELEMENT)=(X-1)*IDIM*IDIM+(Y-1)*IDIM+(Z)
  ELEMENT=ELEMENT+1
ELSE IF (X.EQ.IDIM) THEN
  ROW_MAT%AS(ELEMENT)=ONE
  ROW_MAT%IA1(ELEMENT)=(X-1)*IDIM*IDIM+(Y-1)*IDIM+(Z)
  ELEMENT=ELEMENT+1
ELSE IF (Y.EQ.IDIM) THEN
  ROW_MAT%AS(ELEMENT)=ONE
  ROW_MAT%IA1(ELEMENT)=(X-1)*IDIM*IDIM+(Y-1)*IDIM+(Z)
  ELEMENT=ELEMENT+1
ELSE IF (Z.EQ.IDIM) THEN
  ROW_MAT%AS(ELEMENT)=ONE
  ROW_MAT%IA1(ELEMENT)=(X-1)*IDIM*IDIM+(Y-1)*IDIM+(Z)
  ELEMENT=ELEMENT+1
ELSE
  ! Internal point: build discretization
  !
  ! Term depending on (x-1,y,z)
  !
  ROW_MAT%AS(ELEMENT)=-B1(GLOB_X,GLOB_Y,GLOB_Z)&
    & -A1(GLOB_X,GLOB_Y,GLOB_Z)
  ROW_MAT%AS(ELEMENT) = ROW_MAT%AS(ELEMENT)/(DELTAH*&
    & DELTAH)
  ROW_MAT%IA1(ELEMENT)=(X-2)*IDIM*IDIM+(Y-1)*IDIM+(Z)
  ELEMENT=ELEMENT+1
  ! Term depending on (x,y-1,z)
  ROW_MAT%AS(ELEMENT)=-B2(GLOB_X,GLOB_Y,GLOB_Z)&
    & -A2(GLOB_X,GLOB_Y,GLOB_Z)
  ROW_MAT%AS(ELEMENT) = ROW_MAT%AS(ELEMENT)/(DELTAH*&
    & DELTAH)
  ROW_MAT%IA1(ELEMENT)=(X-1)*IDIM*IDIM+(Y-2)*IDIM+(Z)
  ELEMENT=ELEMENT+1
  ! Term depending on (x,y,z-1)
  ROW_MAT%AS(ELEMENT)=-B3(GLOB_X,GLOB_Y,GLOB_Z)&
    & -A3(GLOB_X,GLOB_Y,GLOB_Z)
  ROW_MAT%AS(ELEMENT) = ROW_MAT%AS(ELEMENT)/(DELTAH*&
    & DELTAH)
  ROW_MAT%IA1(ELEMENT)=(X-1)*IDIM*IDIM+(Y-1)*IDIM+(Z-1)
  ELEMENT=ELEMENT+1
  ! Term depending on (x,y,z)
  ROW_MAT%AS(ELEMENT)=2*B1(GLOB_X,GLOB_Y,GLOB_Z)&
    & +2*B2(GLOB_X,GLOB_Y,GLOB_Z)&
    & +2*B3(GLOB_X,GLOB_Y,GLOB_Z)&
    & +A1(GLOB_X,GLOB_Y,GLOB_Z)&
    & +A2(GLOB_X,GLOB_Y,GLOB_Z)&
    & +A3(GLOB_X,GLOB_Y,GLOB_Z)
  ROW_MAT%AS(ELEMENT) = ROW_MAT%AS(ELEMENT)/(DELTAH*&
    & DELTAH)
  ROW_MAT%IA1(ELEMENT)=(X-1)*IDIM*IDIM+(Y-1)*IDIM+(Z)
  ELEMENT=ELEMENT+1
  ! Term depending on (x,y,z+1)
  ROW_MAT%AS(ELEMENT)=-B1(GLOB_X,GLOB_Y,GLOB_Z)
  ROW_MAT%AS(ELEMENT) = ROW_MAT%AS(ELEMENT)/(DELTAH*&
    & DELTAH)
  ROW_MAT%IA1(ELEMENT)=(X-1)*IDIM*IDIM+(Y-1)*IDIM+(Z+1)
  ELEMENT=ELEMENT+1

```

```

! Term depending on (x,y+1,z)
ROW_MAT%AS(ELEMENT)=-B2(GLOB_X,GLOB_Y,GLOB_Z)
ROW_MAT%AS(ELEMENT) = ROW_MAT%AS(ELEMENT)/(DELTAH*&
& DELTAH)
ROW_MAT%IA1(ELEMENT)=(X-1)*IDIM*IDIM+(Y)*IDIM+(Z)
ELEMENT=ELEMENT+1
! Term depending on (x+1,y,z)
ROW_MAT%AS(ELEMENT)=-B3(GLOB_X,GLOB_Y,GLOB_Z)
ROW_MAT%AS(ELEMENT) = ROW_MAT%AS(ELEMENT)/(DELTAH*&
& DELTAH)
ROW_MAT%IA1(ELEMENT)=(X)*IDIM*IDIM+(Y-1)*IDIM+(Z)
ELEMENT=ELEMENT+1
ENDIF
ROW_MAT%M=1
ROW_MAT%N=N
ROW_MAT%IA2(2)=ELEMENT
! IA== GLOBAL ROW INDEX
IA=(X-1)*IDIM*IDIM+(Y-1)*IDIM+(Z)
T3 = TIMEF()
CALL PSPINS(A,IA,1,ROW_MAT,DESC_A)
TINS = TINS + (TIMEF()-T3)
! Build RHS
IF (X==1) THEN
  GLOB_Y=(Y-IDIM/2)*DELTAH
  GLOB_Z=(Z-IDIM/2)*DELTAH
  ZT(1) = EXP(-GLOB_Y**2-GLOB_Z**2)
ELSE IF ((Y==1).OR.(Y==IDIM).OR.(Z==1).OR.(Z==IDIM)) THEN
  GLOB_X=3*(X-1)*DELTAH
  GLOB_Y=(Y-IDIM/2)*DELTAH
  GLOB_Z=(Z-IDIM/2)*DELTAH
  ZT(1) = EXP(-GLOB_Y**2-GLOB_Z**2)*EXP(-GLOB_X)
ELSE
  ZT(1) = 0.D0
ENDIF
CALL PGEINS(B,ZT(1:1),DESC_A,IA)
ZT(1)=0.D0
CALL PGEINS(T,ZT(1:1),DESC_A,IA)
END IF
END DO
END DO

CALL BLACS_BARRIER(ICONTXT,'ALL')
T2 = TIMEF()

IF (MYPROW.EQ.0) THEN
  WRITE(0,*) ' pspins time',TINS/1.D3
  WRITE(0,*) ' Insert time',(T2-T1)/1.D3
ENDIF

DEALLOCATE(ROW_MAT%AS,ROW_MAT%IA1,ROW_MAT%IA2)

CALL BLACS_BARRIER(ICONTXT,'ALL')
T1 = TIMEF()

CALL PSPASB(A,DESC_A,INFO=INFO,DUPFLAG=0,MTYPE='GEN ')

CALL BLACS_BARRIER(ICONTXT,'ALL')
T2 = TIMEF()

IF (MYPROW.EQ.0) THEN
  WRITE(0,*) ' Assembly time',(T2-T1)/1.D3
ENDIF

CALL PGEASB(B,DESC_A)
CALL PGEASB(T,DESC_A)
RETURN

```

```

END SUBROUTINE CREATE_MATRIX
!
! Functions parameterizing the differential equation
!
FUNCTION A1(X,Y,Z)
  REAL(KIND(1.D0)) :: A1
  REAL(KIND(1.D0)) :: X,Y,Z
  A1=1.D0
END FUNCTION A1
FUNCTION A2(X,Y,Z)
  REAL(KIND(1.D0)) :: A2
  REAL(KIND(1.D0)) :: X,Y,Z
  A2=2.D1*Y
END FUNCTION A2
FUNCTION A3(X,Y,Z)
  REAL(KIND(1.D0)) :: A3
  REAL(KIND(1.D0)) :: X,Y,Z
  A3=1.D0
END FUNCTION A3
FUNCTION A4(X,Y,Z)
  REAL(KIND(1.D0)) :: A4
  REAL(KIND(1.D0)) :: X,Y,Z
  A4=1.D0
END FUNCTION A4
FUNCTION B1(X,Y,Z)
  REAL(KIND(1.D0)) :: B1
  REAL(KIND(1.D0)) :: X,Y,Z
  B1=1.D0
END FUNCTION B1
FUNCTION B2(X,Y,Z)
  REAL(KIND(1.D0)) :: B2
  REAL(KIND(1.D0)) :: X,Y,Z
  B2=1.D0
END FUNCTION B2
FUNCTION B3(X,Y,Z)
  REAL(KIND(1.D0)) :: B3
  REAL(KIND(1.D0)) :: X,Y,Z
  B3=1.D0
END FUNCTION B3
!
! Get iteration parameters from the command line
!
SUBROUTINE GET_PARS(ICONTXT,CMETHD,PREC,IDIM,ISTOPC,ITMAX,ITRACE)
  integer      :: icontxt
  Character*10 :: CMETHD, PREC
  Integer      :: IDIM, IRET, ISTOPC,ITMAX,ITRACE
  Character*40 :: CHARBUF
  INTEGER      :: IARGC, NPROW, NPCOL, MYPROW, MYPCOL
  EXTERNAL     IARGC
  INTEGER      :: INTBUF(10), IP

  CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYPROW, MYPCOL)

  IF (MYPROW==0) THEN
    ! Read command line parameters
    IP=IARGC()
    IF (IARGC().GE.3) THEN
      CALL GETARG(1,CHARBUF)
      READ(CHARBUF,*) CMETHD
      CALL GETARG(2,CHARBUF)
      READ(CHARBUF,*) PREC

      ! Convert strings in array
      DO I = 1, LEN(CMETHD)
        INTBUF(I) = IACHAR(CMETHD(I:I))
      END DO
      ! Broadcast parameters to all processors

```

```

CALL IGEBS2D(ICONTXT,'ALL',' ',10,1,INTBUF,10)

DO I = 1, LEN(PREC)
  INTBUF(I) = IACHAR(PREC(I:I))
END DO
! Broadcast parameters to all processors
CALL IGEBS2D(ICONTXT,'ALL',' ',10,1,INTBUF,10)

CALL GETARG(3,CHARBUF)
READ(CHARBUF,*) IDIM
IF (IARGC().GE.4) THEN
  CALL GETARG(4,CHARBUF)
  READ(CHARBUF,*) ISTOPC
ELSE
  ISTOPC=1
ENDIF
IF (IARGC().GE.5) THEN
  CALL GETARG(5,CHARBUF)
  READ(CHARBUF,*) ITMAX
ELSE
  ITMAX=500
ENDIF
IF (IARGC().GE.6) THEN
  CALL GETARG(6,CHARBUF)
  READ(CHARBUF,*) ITRACE
ELSE
  ITRACE=0
ENDIF
! Broadcast parameters to all processors
CALL IGEBS2D(ICONTXT,'ALL',' ',1,1,IDIM,1)
CALL IGEBS2D(ICONTXT,'ALL',' ',1,1,ISTOPC,1)
CALL IGEBS2D(ICONTXT,'ALL',' ',1,1,ITMAX,1)
CALL IGEBS2D(ICONTXT,'ALL',' ',1,1,ITRACE,1)
WRITE(6,*)'Solving matrix: ELL1'
WRITE(6,*)'on grid',IDIM,'x',IDIM,'x',IDIM
WRITE(6,*)' with BLOCK data distribution, NP=',Np,&
& ' Preconditioner=',PREC,&
& ' Iterative methd=',CMETHD
ELSE
! Wrong number of parameter, print an error message and exit
CALL PR_USAGE(0)
CALL BLACS_ABORT(ICONTXT,-1)
STOP 1
ENDIF
ELSE
! Receive Parameters
CALL IGEBR2D(ICONTXT,'ALL',' ',10,1,INTBUF,10,0,0)
DO I = 1, 10
  CMETHD(I:I) = ACHAR(INTBUF(I))
END DO
CALL IGEBR2D(ICONTXT,'ALL',' ',10,1,INTBUF,10,0,0)
DO I = 1, 10
  PREC(I:I) = ACHAR(INTBUF(I))
END DO
CALL IGEBR2D(ICONTXT,'ALL',' ',1,1,IDIM,1,0,0)
CALL IGEBR2D(ICONTXT,'ALL',' ',1,1,ISTOPC,1,0,0)
CALL IGEBR2D(ICONTXT,'ALL',' ',1,1,ITMAX,1,0,0)
CALL IGEBR2D(ICONTXT,'ALL',' ',1,1,ITRACE,1,0,0)
END IF
RETURN

END SUBROUTINE GET_PARMS
!
! Print an error message
!
SUBROUTINE PR_USAGE(IOUT)
  INTEGER :: IOUT

```

```

WRITE(IOUT,*)'Incorrect parameter(s) found'
WRITE(IOUT,*)' Usage:  pde90 methd prec dim &
    &[istop itmax itrace]'
WRITE(IOUT,*)' Where:'
WRITE(IOUT,*)'      methd:   CGSTAB TFQMR CGS'
WRITE(IOUT,*)'      prec :   ILU DIAGSC NONE'
WRITE(IOUT,*)'      dim      number of points along each axis'
WRITE(IOUT,*)'      the size of the resulting linear '
WRITE(IOUT,*)'      system is dim**3'
WRITE(IOUT,*)'      istop    Stopping criterion  1, 2 or 3 [1]  '
WRITE(IOUT,*)'      itmax    Maximum number of iterations [500] '
WRITE(IOUT,*)'      itrace   0 (no tracing, default) or '
WRITE(IOUT,*)'      >= 0 do tracing every ITRACE'
WRITE(IOUT,*)'      iterations '
END SUBROUTINE PR_USAGE

END PROGRAM PDE90

```

## Fortran 77 Sample Sparse Program

```

C
C This sample program shows how to build and solve a sparse linear
C system using the subroutines in the sparse section of Parallel
C ESSL. The matrix and RHS are generated
C in parallel, so that there is no serial bottleneck.
C
C The program solves a linear system based on the partial differential equation
C
C   b1 dd(u) b2 dd(u) b3 dd(u) a1 d(u) a2 d(u) a3 d(u)
C -  ----- -  ----- -  ----- -  ----- -  ----- -  ----- + a4 u
C     dx dx    dy dy    dz dz      dx      dy      dz
C
C = 0
C
C with Dirichlet boundary conditions on the unit cube
C
C   0<=x,y,z<=1
C
C The equation is discretized with finite differences and uniform stepsize;
C the resulting discrete equation is
C
C ( u(x,y,z) (2b1+2b2+2b3+a1+a2+a3)+u(x-1,y,z) (-b1-a1)+u(x,y-1,z) (-b2-a2)+
C + u(x,y,z-1) (-b3-a3)-u(x+1,y,z) b1-u(x,y+1,z) b2-u(x,y,z+1) b3)*(1/h**2)
C
C In this sample program the index space of the discretized
C computational domain is first numbered sequentially in a standard way,
C then the corresponding vector is distributed according to an HPF BLOCK
C distribution directive.
C
C Boundary conditions are set in a very simple way, by adding
C equations of the form
C
C   u(x,y,z) = rhs(x,y,z)
C
C
C   Program PDE77
C   USE F90SPARSE
C   Implicit none
C
C   EXTERNAL PART_BLOCK
C Input parameters
C   Character*10 :: CMETHD, PREC
C   Integer      :: IDIM
C Miscellaneous
C   Integer, Parameter :: IZERO=0, IONE=1

```

```

Character, PARAMETER :: ORDER='R'
REAL(KIND(1.D0)), POINTER :: B_COL(:), X_COL(:)
INTEGER                :: NR, NNZ, IRCODE, NNZ1, NRHS
REAL(KIND(1.D0)), PARAMETER :: DZERO = 0.D0, ONE = 1.D0
REAL(KIND(1.D0)) :: TIMEF, T1, T2, TPREC, TSOLVE, T3, T4
REAL(KIND(1.D0)), POINTER :: DWORK(:)
EXTERNAL              TIMEF
LOGICAL, PARAMETER :: UPDATE=.TRUE., NOUPDATE=.FALSE.

C Sparse Matrices
REAL(8), POINTER :: AS(:), PRCS(:)
INTEGER, POINTER :: DESC_A(:), IA1(:), IA2(:)
INTEGER          :: INFOA(30)

C Dense Matrices
REAL(KIND(1.D0)), POINTER :: B(:), X(:)
INTEGER                :: LB, LX, LDV, LDV1, IRET

INTERFACE
  SUBROUTINE CREATE_MTRX_ELL1_BLOCK(PARTS, IDIM,
+   AS, IA1, IA2, INFOA, B, T, DESC_A, ICONTXT)
  Implicit None
  external parts
  Integer                :: IDIM
  Real(Kind(1.D0)), Pointer :: B(:), T(:), AS(:)
  integer                :: infoa(30)
  INTEGER, POINTER       :: DESC_A(:), IA1(:), IA2(:)
  Integer                :: ICONTXT
end SUBROUTINE CREATE_MTRX_ELL1_BLOCK
END INTERFACE

C Communications data structure
C BLACS parameters
INTEGER                :: nprow, npcol, icontxt, iam, np, myprow,
+   mypcol

C Solver parameters
Integer                :: iter, itmax, ierr, itrace, methd, iprec,
+   istopc, iparm(20)
real(kind(1.d0))      :: err, eps, rparm(20)

C Other variables
Integer                :: i, info
INTEGER                :: INTERNAL, M, ii, nnzero

C Initialize BLACS
CALL BLACS_PINFO(IAM, NP)
CALL BLACS_GET(IZERO, IZERO, ICONTXT)

C Rectangular Grid, Np x 1

CALL BLACS_GRIDINIT(ICONTXT, ORDER, NP, IONE)
CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYPROW, MYPCOL)

C
C Get parameters
C
CALL GET_PARMS(ICONTXT, CMETHD, PREC, IDIM, ISTOPC, ITMAX, ITRACE)

C
C Allocate and fill in the coefficient matrix and the RHS
C
CALL BLACS_BARRIER(ICONTXT, 'A11')
T1 = TIMEF()
CALL CREATE_MTRX_ELL1_BLOCK(PART_BLOCK, IDIM,
+   AS, IA1, IA2, INFOA, B, X, DESC_A, ICONTXT)
T2 = TIMEF() - T1

```



```

CALL DGAMX2D(ICONTXT,'A',' ',IONE, IONE,T2,IONE,T1,T1,-1,-1,-1)
IF (IAM.EQ.0) Write(6,*) 'Matrix creation Time : ',T2/1.D3

LB  = SIZE(B)
LX  = SIZE(X)
LDV = DESC_A(5)
LDV1 = DESC_A(6)
NNZ = SIZE(AS)
NNZ1 = SIZE(IA1)

ALLOCATE(P RCS(10+2*NNZ+LDV+LDV1+40),STAT=IRCODE)
IF (IRCODE /= 0) THEN
    WRITE(0,*) 'Allocation error'
    CALL BLACS_ABORT(ICONTXT,-1)
    STOP
ENDIF

C
C Prepare the preconditioning data structure
C
    SELECT CASE (PREC)
    CASE ('ILU')
        IPREC = 2
    CASE ('DIAGSC')
        IPREC = 1
    CASE ('NONE')
        IPREC = 0
    CASE DEFAULT
        WRITE(0,*) 'Unknown preconditioner'
        CALL BLACS_ABORT(ICONTXT,-1)
    END SELECT

    CALL BLACS_BARRIER(ICONTXT,'All')
    T1 = TIMEF()
    CALL PDSPGPR(IPREC,AS,IA1,IA2,INFOA,P RCS,SIZE(P RCS),DESC_A,IRET)
    TPREC = TIMEF()-T1

    CALL DGAMX2D(icontxt,'A',' ',IONE, IONE,TPREC,IONE,t1,t1,-1,-1,-1)

    IF (IAM.EQ.0) WRITE(6,*) 'Preconditioner Time : ',TPREC/1.D3
    if (iret.ne.0) then
        write(0,*) 'Error on preconditioner',iret
        call blacs_abort(icontxt,-1)
        stop
    endif

C
C Iteration parameters
C
    IF (CMETHD(1:6).EQ.'CGSTAB') Then
        METHD = 1
    ELSE IF (CMETHD(1:3).EQ.'CGS') Then
        METHD = 2
    ELSE IF (CMETHD(1:5).EQ.'TFQMR') THEN
        METHD = 3
    ELSE
        WRITE(0,*) 'Unknown method '
        CALL BLACS_ABORT(ICONTXT,-1)
    END IF
    EPS = 1.D-9

    IPARM = 0
    RPARM = 0.D0
    IPARM(1) = METHD
    IPARM(2) = ISTOPC

```

```

IPARM(3) = ITMAX
IPARM(4) = ITRACE
RPARM(1) = EPS

NRHS = 1

CALL BLACS_BARRIER(ICONTXT,'All')
T1 = TIMEF()
CALL PDSPGIS(AS,IA1,IA2,INFOA,NRHS,B,LB,X,LX,PRCS,
+  DESC_A,IPARM,RPARM,INFO)
CALL BLACS_BARRIER(ICONTXT,'All')
TSOLVE = TIMEF() - T1
ERR      = RPARM(2)
ITER     = IPARM(5)

IF (IAM.EQ.0) THEN
  WRITE(6,*) 'Time to Solve Matrix : ',TSOLVE/1.D3
  WRITE(6,*) 'Time per iteration : ',TSOLVE/(1.D3*ITER)
  WRITE(6,*) 'Number of iterations : ',ITER
  WRITE(6,*) 'Error on exit : ',ERR
  WRITE(6,*) 'INFO on exit: ',INFO
END IF

CALL BLACS_GRIDEXIT(ICONTXT)
CALL BLACS_EXIT(0)
STOP

END

C
C Print an error message
C
SUBROUTINE PR_USAGE(IOUT)
  INTEGER :: IOUT
  WRITE(IOUT,*) 'Incorrect parameter(s) found'
  WRITE(IOUT,*)
+  ' Usage: pde77 methd prec dim [istopc itmax itrace]'
  WRITE(IOUT,*) ' Where:'
  WRITE(IOUT,*) '      methd:   CGSTAB TFQMR CGS'
  WRITE(IOUT,*) '      prec :   ILU DIAGSC NONE'
  WRITE(IOUT,*) '      dim      number of points along each axis'
  WRITE(IOUT,*) '              the size of the resulting linear '
  WRITE(IOUT,*) '              system is dim**3'
  WRITE(IOUT,*) '      istopc  Stopping criterion 1 2 or 3 [1] '
  WRITE(IOUT,*) '      itmax    Maximum number of iterations [500]'
  WRITE(IOUT,*) '      itrace    0 (no tracing, default) or '
  WRITE(IOUT,*) '              >= 0 do tracing every ITRACE'
  WRITE(IOUT,*) '              iterations '
  RETURN
END

C
C Functions parameterizing the differential equation
C
FUNCTION A1(X,Y,Z)
  REAL(KIND(1.D0)) :: A1
  REAL(KIND(1.D0)) :: X,Y,Z
  A1=1.D0
END
FUNCTION A2(X,Y,Z)
  REAL(KIND(1.D0)) :: A2
  REAL(KIND(1.D0)) :: X,Y,Z

  A2=2.D1*Y
END
FUNCTION A3(X,Y,Z)
  REAL(KIND(1.D0)) :: A3

```

```

REAL(KIND(1.D0)) :: X,Y,Z
A3=1.D0
END
FUNCTION A4(X,Y,Z)
REAL(KIND(1.D0)) :: A4
REAL(KIND(1.D0)) :: X,Y,Z
A4=1.D0
END
FUNCTION B1(X,Y,Z)
REAL(KIND(1.D0)) :: B1
REAL(KIND(1.D0)) :: X,Y,Z
B1=1.D0
END
FUNCTION B2(X,Y,Z)
REAL(KIND(1.D0)) :: B2
REAL(KIND(1.D0)) :: X,Y,Z
B2=1.D0
END
FUNCTION B3(X,Y,Z)
REAL(KIND(1.D0)) :: B3
REAL(KIND(1.D0)) :: X,Y,Z
B3=1.D0
END

C
C Subroutine to allocate and fill in the coefficient matrix and
C the RHS.
C
      SUBROUTINE CREATE_MTRX_ELL1_BLOCK(PARTS,IDIM,
+    AS,IA1,IA2,INFOA,B,T,DESC_A,ICONTXT)
C
C   the equation generated is:
C   b1 d d (u)  b2 d d (u)  b3 d d (u)  a1 d (u)  a2 d (u))  a3d (u))  a4 u
C -  ----- -  ----- -  ----- -  ----- -  ----- -  ----- +
C    dx dx      dy dy      dz dz      dx      dy      dz
C
C =g(x,y,z)
C where g is the RHS extracted from exact solution:
C f(x,y,z)=10.d0*X*Y*Z*(1-X)*(1-Y)*(1-Z)*EXP(X**4.5)
C boundary condition: Dirichlet
C 0< x,y,z<1
C discretized with finite differences; the discrete equation is
C u(x,y,z) (2b1+2b2+2b3+a1+a2+a3)+u(x-1,y,z) (-b1-a1)+u(x,y-1,z) (-b2-a2)+
C + u(x,y,z-1) (-b3-a3)-u(x+1,y,z)b1-u(x,y+1,z)b2-u(x,y,z+1)b3
C !!this matrix is non symmetric
      USE F90SPARSE
      EXTERNAL PARTS

      Implicit None
      Integer :: IDIM
      Real(Kind(1.D0)),Pointer :: B(:), T(:), AS(:)
      integer :: infoa(20)
      INTEGER, POINTER :: DESC_A(:), IA1(:),IA2(:)
      Integer :: ICONTXT
      Real(Kind(1.d0)) :: ZT(10),GLOB_X,GLOB_Y,GLOB_Z,
+    ras(20)
      Integer :: M,N,NNZ,glob_row,nr,j
      integer :: rial(20),ria2(20),rinfoa(30)
      Real(Kind(1.D0)),POINTER :: SOL(:)
      real(kind(1.d0)), external :: a1,a2,a3,b1,b2,b3
      Integer :: X,Y,Z,COUNTER,IA,I,NPROW,NPCOL,MYPROW
+    ,MYPCOL,DOMAIN_INDEX
      Integer :: BOUND_COND_0YZ, BOUND_COND_1YZ,
+    BOUND_COND_X0Z , BOUND_COND_X1Z , BOUND_COND_XY0 ,
+    BOUND_COND_XY1,MP,ELEMENT,LDSCA,IRCODE, NNZ1
      REAL(KIND(1.D0)) :: DELTAH
      INTEGER :: GAP,INFO

```

```

        integer      :: prv(64), indx_owner, nv, inv
C deltax dimension of each grid cell
C deltat discretization time
        Real(Kind(1.d0)),Parameter  :: RHS=0.d0,ONE=1.d0,ZERO=0.d0
        Real(Kind(1.d0))  :: TIMEF, T1, T2,t3, TINS
        external          timef
C common area
        INTEGER DIM_BLOCK, NPROC

        CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYPROW, MYPCOL)

        NPROC = NPROW*NPCOL
        DELTAH=1.D0/MAX((IDIM-1),1)

        M   = IDIM*IDIM*IDIM
        N   = M
        LDSCA = 3*N+31+3*NPROC
        DIM_BLOCK=(N+NPROC-1)/NPROC
        NNZ   = MAX(2,(N*7)/(NPROC))
        NNZ1  = MAX(2,(N*9)/(NPROC))+MAX(1,DIM_BLOCK)
        ALLOCATE(DESC_A(LDSCA),AS(NNZ),IA1(NNZ1),
+   IA2(NNZ1),STAT=IRCODE)
        IF (IRCODE /= 0) THEN
            WRITE(0,*) 'Allocation error in CREATE'
            CALL BLACS_ABORT(ICONTXT,-1)
            STOP
        ENDIF
        INFOA(1) = NNZ
        INFOA(2) = NNZ1
        INFOA(3) = NNZ1

        DESC_A(11) = LDSCA
        CALL PADINIT(N,PARTS,DESC_A,ICONTXT)

        NR = MAX(DESC_A(5),1)
        ALLOCATE(B(NR),T(NR),STAT=IRCODE)
        IF (IRCODE /= 0) THEN
            WRITE(0,*) 'Allocation error in CREATE'
            CALL BLACS_ABORT(ICONTXT,-1)
            STOP
        ENDIF

        CALL PDSPINIT(AS,IA1,IA2,INFOA,DESC_A)

C
C We build an auxiliary matrix consisting of one row at a
C time in CSR mode
C
        RINFOA(4) = 1
        RINFOA(5) = 1
        RINFOA(6) = 1
        RINFOA(7) = N

        GAP = 1
        RIA2(1)=1
        TINS = 0.D0

        CALL BLACS_BARRIER(ICONTXT,'ALL')
        T1 = TIMEF()
C Loop over all rows which belongs to me; we have a BLOCK
C distribution !!
        DO GLOB_ROW = 1, N
            CALL PARTS(GLOB_ROW,N,NPROW,PRV,NV)
            DO INV = 1, NV
                INDX_OWNER = PRV(INV)

```

```

        IF (INDX_OWNER == MYPROW) THEN
            ELEMENT=1
C        GLOB_X, GLOB_Y, GLOB_Z coordinates in current measure unit

C Compute Point Coordinates
        IF (MOD(GLOB_ROW,(IDIM*IDIM)).EQ.0) THEN
            X = GLOB_ROW/(IDIM*IDIM)
        ELSE
            X = GLOB_ROW/(IDIM*IDIM)+1
        ENDIF
        IF (MOD((GLOB_ROW-(X-1)*IDIM*IDIM),IDIM).EQ.0) THEN
            Y = (GLOB_ROW-(X-1)*IDIM*IDIM)/IDIM
        ELSE
            Y = (GLOB_ROW-(X-1)*IDIM*IDIM)/IDIM+1
        ENDIF
        Z = GLOB_ROW-(X-1)*IDIM*IDIM-(Y-1)*IDIM
        GLOB_X=X*DELTAH
        GLOB_Y=Y*DELTAH
        GLOB_Z=Z*DELTAH
        IF (X.EQ.1) THEN
            RAS(ELEMENT)=ONE
            RIA1(ELEMENT)=(X-1)*IDIM*IDIM+(Y-1)*IDIM+(Z)
            ELEMENT=ELEMENT+1
        ELSE IF (Y.EQ.1) THEN
            RAS(ELEMENT)=ONE
            RIA1(ELEMENT)=(X-1)*IDIM*IDIM+(Y-1)*IDIM+(Z)
            ELEMENT=ELEMENT+1
        ELSE IF (Z.EQ.1) THEN
            RAS(ELEMENT)=ONE
            RIA1(ELEMENT)=(X-1)*IDIM*IDIM+(Y-1)*IDIM+(Z)
            ELEMENT=ELEMENT+1
        ELSE IF (X.EQ.IDIM) THEN
            RAS(ELEMENT)=ONE
            RIA1(ELEMENT)=(X-1)*IDIM*IDIM+(Y-1)*IDIM+(Z)
            ELEMENT=ELEMENT+1
        ELSE IF (Y.EQ.IDIM) THEN
            RAS(ELEMENT)=ONE
            RIA1(ELEMENT)=(X-1)*IDIM*IDIM+(Y-1)*IDIM+(Z)
            ELEMENT=ELEMENT+1
        ELSE IF (Z.EQ.IDIM) THEN
            RAS(ELEMENT)=ONE
            RIA1(ELEMENT)=(X-1)*IDIM*IDIM+(Y-1)*IDIM+(Z)
            ELEMENT=ELEMENT+1
        ELSE
C        !      ....internal point.....
C        !      (x-1,y,z)
            RAS(ELEMENT)=-B1(GLOB_X,GLOB_Y,GLOB_Z)
            -A1(GLOB_X,GLOB_Y,GLOB_Z)
            +      RAS(ELEMENT) = RAS(ELEMENT)/(DELTAH*DELTAH)
            RIA1(ELEMENT)=(X-2)*IDIM*IDIM+(Y-1)*IDIM+(Z)
            ELEMENT=ELEMENT+1
C        !      (x,y-1,z)
            RAS(ELEMENT)=-B2(GLOB_X,GLOB_Y,GLOB_Z)
            -A2(GLOB_X,GLOB_Y,GLOB_Z)
            +      RAS(ELEMENT) = RAS(ELEMENT)/(DELTAH*DELTAH)
            RIA1(ELEMENT)=(X-1)*IDIM*IDIM+(Y-2)*IDIM+(Z)
            ELEMENT=ELEMENT+1
C        !      (x,y,z-1)
            RAS(ELEMENT)=-B3(GLOB_X,GLOB_Y,GLOB_Z)
            -A3(GLOB_X,GLOB_Y,GLOB_Z)
            +      RAS(ELEMENT) = RAS(ELEMENT)/(DELTAH*DELTAH)
            RIA1(ELEMENT)=(X-1)*IDIM*IDIM+(Y-1)*IDIM+(Z-1)
            ELEMENT=ELEMENT+1
C        !      (x,y,z)
            RAS(ELEMENT)=2*B1(GLOB_X,GLOB_Y,GLOB_Z)
            +      +2*B2(GLOB_X,GLOB_Y,GLOB_Z)
            +      +2*B3(GLOB_X,GLOB_Y,GLOB_Z)

```

```

+          +A1(GLOB_X,GLOB_Y,GLOB_Z)
+          +A2(GLOB_X,GLOB_Y,GLOB_Z)
+          +A3(GLOB_X,GLOB_Y,GLOB_Z)
          RAS(ELEMENT) = RAS(ELEMENT)/(DELTAH*DELTAH)
          RIA1(ELEMENT)=(X-1)*IDIM*IDIM+(Y-1)*IDIM+(Z)
          ELEMENT=ELEMENT+1
C          !
          (x,y,z+1)
          RAS(ELEMENT)=-B1(GLOB_X,GLOB_Y,GLOB_Z)
          RAS(ELEMENT) = RAS(ELEMENT)/(DELTAH*DELTAH)
          RIA1(ELEMENT)=(X-1)*IDIM*IDIM+(Y-1)*IDIM+(Z+1)
          ELEMENT=ELEMENT+1
C          !
          (x,y+1,z)
          RAS(ELEMENT)=-B2(GLOB_X,GLOB_Y,GLOB_Z)
          RAS(ELEMENT) = RAS(ELEMENT)/(DELTAH*DELTAH)
          RIA1(ELEMENT)=(X-1)*IDIM*IDIM+(Y)*IDIM+(Z)
          ELEMENT=ELEMENT+1
C          !
          (x+1,y,z)
          RAS(ELEMENT)=-B3(GLOB_X,GLOB_Y,GLOB_Z)
          RAS(ELEMENT) = RAS(ELEMENT)/(DELTAH*DELTAH)
          RIA1(ELEMENT)=(X)*IDIM*IDIM+(Y-1)*IDIM+(Z)
          ELEMENT=ELEMENT+1
          ENDIF
          RIA2(2) = ELEMENT
          RINFOA(1) = 20
          RINFOA(2) = 20
          RINFOA(3) = 20
          RINFOA(4) = 1
          RINFOA(5) = 1
          RINFOA(6) = 1
C IA== GLOBAL ROW INDEX
          IA=(X-1)*IDIM*IDIM+(Y-1)*IDIM+(Z)
          T3 = TIMEF()
          CALL PDSPINS(AS,IA1,IA2,INFOA,DESC_A,
+          IA,1,RAS,RIA1,RIA2,RINFOA)
          TINS = TINS + (TIMEF()-T3)
C Build RHS
          IF (X==1) THEN
            GLOB_Y=(Y-IDIM/2)*DELTAH
            GLOB_Z=(Z-IDIM/2)*DELTAH
            ZT(1) = EXP(-GLOB_Y**2-GLOB_Z**2)
          ELSE IF ((Y==1).OR.(Y==IDIM).OR.(Z==1).OR.(Z==IDIM)) THEN
            GLOB_X=3*(X-1)*DELTAH
            GLOB_Y=(Y-IDIM/2)*DELTAH
            GLOB_Z=(Z-IDIM/2)*DELTAH
            ZT(1) = EXP(-GLOB_Y**2-GLOB_Z**2)*EXP(-GLOB_X)
          ELSE
            ZT(1) = 0.D0
          ENDIF
          CALL PDGEINS(1,B,NR,IA,1,1,1,ZT,1,DESC_A)
          ZT(1) = 0.D0
          CALL PDGEINS(1,T,NR,IA,1,1,1,ZT,1,DESC_A)
          ENDIF
        ENDDO
      ENDDO

      CALL BLACS_BARRIER(ICONTXT,'ALL')
      T2 = TIMEF()

      IF (MYPROW.EQ.0) THEN
        WRITE(0,*) '      pspins time',TINS/1.D3
        WRITE(0,*) '      Insert time',(T2-T1)/1.D3
      ENDIF

      CALL BLACS_BARRIER(ICONTXT,'ALL')
      T1 = TIMEF()

      CALL PDSPASB(AS,IA1,IA2,INFOA,DESC_A,

```

```

+ 'GEN ', 'DEF ', 0, INFO)

CALL BLACS_BARRIER(ICONTXT, 'ALL')
T2 = TIMEF()

IF (MYPROW.EQ.0) THEN
  WRITE(0,*) ' Assembly time', (T2-T1)/1.D3
ENDIF

CALL PDGEASB(1,B,NR,DESC_A)
CALL PDGEASB(1,T,NR,DESC_A)
RETURN
END

C
C Get iteration parameters from the command line
C
  SUBROUTINE GET_PARMS(ICONTXT, CMETHD, PREC, IDIM,
+   ISTOPC, ITMAX, ITRACE)
  integer      :: icontxt
  Character*10 :: CMETHD, PREC
  Integer      :: IDIM, IRET, ISTOPC, ITMAX, ITRACE
  Character*40 :: CHARBUF
  INTEGER      :: IARGC, NPROW, NPCOL, MYPROW, MYPCOL
  EXTERNAL     IARGC
  INTEGER      :: INTBUF(10), IP

  CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYPROW, MYPCOL)

  IF (MYPROW==0) THEN
C Read command line parameters
    IP=IARGC()
    IF (IARGC().GE.3) THEN
      CALL GETARG(1, CHARBUF)
      READ(CHARBUF,*) CMETHD
      CALL GETARG(2, CHARBUF)
      READ(CHARBUF,*) PREC

C Convert strings in array
      DO I = 1, LEN(CMETHD)
        INTBUF(I) = IACHAR(CMETHD(I:I))
      END DO
C Broadcast parameters to all processors
      CALL IGEBS2D(ICONTXT, 'ALL', ' ', 10, 1, INTBUF, 10)

      DO I = 1, LEN(PREC)
        INTBUF(I) = IACHAR(PREC(I:I))
      END DO
C Broadcast parameters to all processors
      CALL IGEBS2D(ICONTXT, 'ALL', ' ', 10, 1, INTBUF, 10)

      CALL GETARG(3, CHARBUF)
      READ(CHARBUF,*) IDIM
      IF (IARGC().GE.4) THEN
        CALL GETARG(4, CHARBUF)
        READ(CHARBUF,*) ISTOPC
      ELSE
        ISTOPC=1
      ENDIF
      IF (IARGC().GE.5) THEN
        CALL GETARG(5, CHARBUF)
        READ(CHARBUF,*) ITMAX
      ELSE
        ITMAX=500
      ENDIF
      IF (IARGC().GE.6) THEN
        CALL GETARG(6, CHARBUF)
        READ(CHARBUF,*) ITRACE

```

```

        ELSE
            ITRACE=0
        ENDIF
C Broadcast parameters to all processors
        CALL IGEBS2D(ICONXT,'ALL',' ',1,1,IDIM,1)
        CALL IGEBS2D(ICONXT,'ALL',' ',1,1,ISTOPC,1)
        CALL IGEBS2D(ICONXT,'ALL',' ',1,1,ITMAX,1)
        CALL IGEBS2D(ICONXT,'ALL',' ',1,1,ITRACE,1)
        WRITE(6,*)'Solving matrix: ELL1'
        WRITE(6,*)'on grid',IDIM,'x',IDIM,'x',IDIM
        WRITE(6,*)' with BLOCK data distribution, NP=',Np,
+           ' Preconditioner=',PREC,
+           ' Iterative methd=',CMETHD
        ELSE
C Wrong number of parameter, print an error message and exit
        CALL PR_USAGE(0)
        CALL BLACS_ABORT(ICONXT,-1)
        STOP 1
    ENDIF
ELSE
C Receive Parameters
        CALL IGEBR2D(ICONXT,'ALL',' ',10,1,INTBUF,10,0,0)
        DO I = 1, 10
            CMETHD(I:I) = ACHAR(INTBUF(I))
        END DO
        CALL IGEBR2D(ICONXT,'ALL',' ',10,1,INTBUF,10,0,0)
        DO I = 1, 10
            PREC(I:I) = ACHAR(INTBUF(I))
        END DO
        CALL IGEBR2D(ICONXT,'ALL',' ',1,1,IDIM,1,0,0)
        CALL IGEBR2D(ICONXT,'ALL',' ',1,1,ISTOPC,1,0,0)
        CALL IGEBR2D(ICONXT,'ALL',' ',1,1,ITMAX,1,0,0)
        CALL IGEBR2D(ICONXT,'ALL',' ',1,1,ITRACE,1,0,0)
    END IF
    RETURN
END

```

## Fortran 90 Sample Sparse Program (using the Harwell-Boeing exchange format)

```

@PROCESS FREE(F90) INIT(F90PTR)
!
! This sample program shows how to build and solve a sparse linear
! system using the subroutines in the sparse section of Parallel
! ESSL; the matrices are read from file using the Harwell-Boeing
! exchange format. Details on the format and sample matrices are
! available from
!
! http://math.nist.gov/MatrixMarket/
!
! The user can choose between different data distribution strategies.
! These are equivalents to the HPF BLOCK and CYCLIC(N) distributions;
! they do not take into account the sparsity pattern of the input
! matrix.
!
PROGRAM HB_SAMPLE
    USE F90SPARSE
    USE MAT_DIST
    USE READ_MAT
    USE PARTRAND
    USE PARTBCYC
    IMPLICIT NONE

    ! Input parameters
    CHARACTER*40 :: CMETHD, PREC, MTRX_FILE

```



```

CHARACTER*80 :: CHARBUF

DOUBLE PRECISION DDOT
EXTERNAL DDOT
EXTERNAL PART_BLOCK

INTEGER, PARAMETER :: IZERO=0, IONE=1
CHARACTER, PARAMETER :: ORDER='R'
REAL(KIND(1.D0)), POINTER,SAVE :: B_COL(:), X_COL(:), R_COL(:), &
    & B_COL_GLOB(:), X_COL_GLOB(:), R_COL_GLOB(:), B_GLOB(:,:)
INTEGER :: IARGC
Real(Kind(1.d0)), Parameter :: Dzero = 0.d0, One = 1.d0
Real(Kind(1.d0)) :: TIMEF, T1, T2, TPREC, R_AMAX, B_AMAX,bb(1,1)
integer :: nrhs, nrow, nx1, nx2
External IARGC, TIMEF
integer bsize,overlap
common/part/bsize,overlap

! Sparse Matrices
TYPE(D_SPMAT) :: A, AUX_A
TYPE(D_PRECN) :: APRC
! Dense Matrices
REAL(KIND(1.D0)), POINTER :: AUX_B(:,:) , AUX1(:), AUX2(:)

! Communications data structure
TYPE(DESC_TYPE) :: DESC_A

! BLACS parameters
INTEGER :: NPROW, NPCOL, ICTXT, IAM, NP, MYPROW, MYPCOL

! Solver paramters
INTEGER :: ITER, ITMAX, IERR, ITRACE, IRCODE, IPART,&
    & IPREC, METHD, ISTOPC
REAL(KIND(1.D0)) :: ERR, EPS
integer iparm(20)
real(kind(1.d0)) rparm(20)

! Other variables
INTEGER :: I,INFO,J
INTEGER :: INTERNAL, M,II,NNZERO

! common area
INTEGER M_PROBLEM, NPROC

! Initialize BLACS
CALL BLACS_PINFO(IAM, NP)
CALL BLACS_GET(IZERO, IZERO, ICTXT)

! Rectangular Grid, Np x 1

CALL BLACS_GRIDINIT(ICTXT, ORDER, NP, IONE)
CALL BLACS_GRIDINFO(ICTXT, NPROW, NPCOL, MYPROW, MYPCOL)

!
! Get parameters
!
CALL GET_PARMs(ICTXT,MTRX_FILE,CMETHD,PREC,&
    & IPART,ISTOPC,ITMAX,ITRACE)

CALL BLACS_BARRIER(ICTXT,'A')
T1 = TIMEF()
! Read the input matrix to be processed and (possibly) the RHS
IF (IAM == 0) THEN
    CALL READMAT(MTRX_FILE, AUX_A, ICTXT,B=AUX_B)
    M_PROBLEM = AUX_A%M
    CALL IGBS2D(ICTXT,'A',' ',1,1,M_PROBLEM,1)
    IF (SIZE(AUX_B,1).EQ.M_PROBLEM) THEN

```

```

        ! If any RHS were present, broadcast the first one
        NRHS = 1
        CALL IGEBS2D(ICTXT,'A',' ',1,1,NRHS,1)
        CALL DGEBS2D(ICTXT,'A',' ',M_PROBLEM,1,AUX_B(:,1),M_PROBLEM)
    ELSE
        NRHS = 0
        CALL IGEBS2D(ICTXT,'A',' ',1,1,NRHS,1)
    ENDIF
ELSE
    CALL IGEBR2D(ICTXT,'A',' ',1,1,M_PROBLEM,1,0,0)
    CALL IGEBR2D(ICTXT,'A',' ',1,1,NRHS,1,0,0)
    IF (NRHS.EQ.1) THEN
        ALLOCATE(AUX_B(M_PROBLEM,1), STAT=IRCODE)
        IF (IRCODE /= 0) THEN
            WRITE(0,*) 'Memory allocation failure in HB_SAMPLE'
            CALL BLACS_ABORT(ICTXT,-1)
            STOP
        ENDIF
        CALL DGEBR2D(ICTXT,'A',' ',M_PROBLEM,1,AUX_B,M_PROBLEM,0,0)
    ENDIF
END IF
IF (NRHS.EQ.1) THEN
    B_COL_GLOB =>AUX_B(:,1)
ELSE
    ALLOCATE(AUX_B(M_PROBLEM,1), STAT=IRCODE)
    B_COL_GLOB =>AUX_B(:,1)
    IF (IAM==0) THEN
        DO I=1, M_PROBLEM
            B_COL_GLOB(I) = REAL(I)*2.0/REAL(M_PROBLEM)
        ENDDO
    ENDIF
ENDIF
ENDIF
NPROC = NPROW

! Switch over different partition types
IF (IPART > 0) THEN
    WRITE(6,*) 'Partition type: CYCLIC(NB)'
    CALL SET_NB(IPART,0,0,ICTXT)
    CALL MATDIST(AUX_A, A, PART_BCYC, ICTXT, &
        & DESC_A,B_COL_GLOB,B_COL)
ELSE
    SELECT CASE (IPART)

    CASE (0)
        WRITE(6,*) 'Partition type: BLOCK'
        CALL MATDIST(AUX_A, A, PART_BLOCK, ICTXT, &
            & DESC_A,B_COL_GLOB,B_COL)
    CASE (-1)
        WRITE(6,*) 'Partition type: RANDOM'
        IF (IAM==0) THEN
            CALL BUILD_RNDPART(AUX_A,NP)
        ENDIF
        CALL DISTR_RNDPART(0,0,ICTXT)
        CALL MATDIST(AUX_A, A, PART_RAND, ICTXT, &
            & DESC_A,B_COL_GLOB,B_COL)
    CASE DEFAULT
        WRITE(6,*) 'Partition type: BLOCK'
        CALL MATDIST(AUX_A, A, PART_BLOCK, ICTXT, &
            & DESC_A,B_COL_GLOB,B_COL)
    END SELECT
ENDIF

CALL PGEALL(X_COL,DESC_A)
CALL PGEASB(X_COL,DESC_A)
T2 = TIMEF() - T1

```

```

CALL DGAMX2D(ICTXT, 'A', ' ', IONE, IONE, T2, IONE,&
& T1, T1, -1, -1, -1)

IF (IAM.EQ.0) THEN
  WRITE(6,*) 'Time to Read and Partition Matrix : ',T2/1.D3
END IF

!
! Prepare the preconditioning matrix. Note the availability
! of optional parameters
!
IF (PREC(1:3) == 'ILU') THEN
  IPREC = 2
ELSE IF (PREC(1:6) == 'DIAGSC') THEN
  IPREC = 1
ELSE IF (PREC(1:4) == 'NONE') THEN
  IPREC = 0
ELSE
  WRITE(0,*) 'Unknown preconditioner'
  CALL BLACS_ABORT(ICTXT,-1)
END IF
CALL BLACS_BARRIER(ICTXT,'A')
T1 = TIMEF()
CALL PSPGPR(IPREC,A,APRC,DESC_A,INFO=INFO)
TPREC = TIMEF()-T1

CALL DGAMX2D(ICTXT,'A',' ', IONE, IONE,TPREC,IONE,T1,T1,-1,-1,-1)

IF (IAM.EQ.0) WRITE(6,*) 'Preconditioner Time : ',TPREC/1.D3
IF (INFO /= 0) THEN
  WRITE(0,*) 'Error in preconditioner : ',INFO
  CALL BLACS_ABORT(ICTXT,-1)
  STOP
END IF

IPARM = 0
RPARM = 0.D0

EPS = 1.D-8
RPARM(1) = EPS
IPARM(2) = ISTOPC
IPARM(3) = ITMAX
IPARM(4) = ITRACE
IF (CMETHD(1:6).EQ.'CGSTAB') Then
  IPARM(1)=1
ELSE IF (CMETHD(1:3).EQ.'CGS') THEN
  IPARM(1)=2
ELSE IF (CMETHD(1:5).EQ.'TFQMR') THEN
  IPARM(1)=3
ELSE
  WRITE(0,*) 'Unknown method '
  CALL BLACS_ABORT(ICTXT,-1)
END IF

CALL BLACS_BARRIER(ICTXT,'All')
T1 = TIMEF()
CALL PSPGIS(A,B_COL,X_COL,APRC,DESC_A,&
& IPARM=IPARM,RPARM=RPARM,INFO=IERR)
CALL BLACS_BARRIER(ICTXT,'All')
T2 = TIMEF() - T1
CALL DGAMX2D(ICTXT,'A',' ', IONE, IONE,T2,IONE,T1,T1,-1,-1,-1)
ITER=IPARM(5)
ERR = RPARM(2)
IF (IAM.EQ.0) THEN

```

```

        WRITE(6,*) 'methd iprec istopc  : ',METHD, IPREC, ISTOPC
        WRITE(6,*) 'Number of iterations : ',ITER
        WRITE(6,*) 'Time to Solve Matrix : ',T2/1.D3
        WRITE(6,*) 'Time per iteration  : ',T2/(1.D3*ITER)
        WRITE(6,*) 'Error on exit      : ',ERR
    END IF

    CALL PGEFREE(B_COL, DESC_A)
    CALL PGEFREE(X_COL, DESC_A)
    CALL PSPFREE(A, DESC_A)
    CALL PSPFREE(APRC, DESC_A)
    CALL PADFREE(DESC_A)
    CALL BLACS_GRIDEXIT(ICTXT)
    CALL BLACS_EXIT(0)

CONTAINS
!
! Get iteration parameters from the command line
!
SUBROUTINE GET_PARMS(ICONTXT,MTRX_FILE,CMETHD,PREC,IPART,&
    & ISTOPC,ITMAX,ITRACE)
    integer      :: icontxt
    Character*40 :: CMETHD, PREC, MTRX_FILE
    Integer      :: IRET, ISTOPC,ITMAX,ITRACE,IPART
    Character*40 :: CHARBUF
    INTEGER      :: IARGC, NPROW, NPCOL, MYPROW, MYPCOL
    EXTERNAL     IARGC
    INTEGER      :: INPARMS(20), IP

    CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYPROW, MYPCOL)
    IF (MYPROW==0) THEN
        ! Read Input Parameters
        IF (IARGC().GE.3) THEN
            CALL GETARG(1,CHARBUF)
            READ(CHARBUF,*) MTRX_FILE
            CALL GETARG(2,CHARBUF)
            READ(CHARBUF,*) CMETHD
            CALL GETARG(3,CHARBUF)
            READ(CHARBUF,*) PREC
            IF (IARGC().GE.4) THEN
                CALL GETARG(4,CHARBUF)
                READ(CHARBUF,*) IPART
            ELSE
                IPART = 0
            ENDIF
            IF (IARGC().GE.5) THEN
                CALL GETARG(5,CHARBUF)
                READ(CHARBUF,*) ITMAX
            ELSE
                ITMAX = 500
            ENDIF
            IF (IARGC().GE.6) THEN
                CALL GETARG(6,CHARBUF)
                READ(CHARBUF,*) ISTOPC
            ELSE
                ISTOPC = 1
            ENDIF
            IF (IARGC().GE.7) THEN
                CALL GETARG(7,CHARBUF)
                READ(CHARBUF,*) ITRACE
            ELSE
                ITRACE = 0
            ENDIF

            ! Convert strings to integers
            DO I = 1, 20
                INPARMS(I) = IACHAR(MTRX_FILE(I:I))
            END DO
        END IF
    END IF

```

```

        END DO
        ! Broadcast parameters to all processors
        CALL IGEBS2D(ICTXT,'A',' ',20,1,INPARMS,20)

        ! Convert strings in array
        DO I = 1, 20
            INPARMS(I) = IACHAR(CMETHD(I:I))
        END DO
        ! Broadcast parameters to all processors
        CALL IGEBS2D(ICTXT,'A',' ',20,1,INPARMS,20)

        DO I = 1, 20
            INPARMS(I) = IACHAR(PREC(I:I))
        END DO
        ! Broadcast parameters to all processors
        CALL IGEBS2D(ICTXT,'A',' ',20,1,INPARMS,20)

        ! Broadcast parameters to all processors
        CALL IGEBS2D(ICTXT,'A',' ',1,1,IPART,1)
        CALL IGEBS2D(ICTXT,'A',' ',1,1,ITMAX,1)
        CALL IGEBS2D(ICTXT,'A',' ',1,1,ISTOPC,1)
        CALL IGEBS2D(ICTXT,'A',' ',1,1,ITRACE,1)

    ELSE
        CALL PR_USAGE(0)
        CALL BLACS_ABORT(ICTXT,-1)
        STOP 1
    END IF
ELSE
    ! Receive Parameters
    CALL IGEBR2D(ICTXT,'A',' ',20,1,INPARMS,20,0,0)
    DO I = 1, 20
        MTRX_FILE(I:I) = ACHAR(INPARMS(I))
    END DO

    CALL IGEBR2D(ICTXT,'A',' ',20,1,INPARMS,20,0,0)
    DO I = 1, 20
        CMETHD(I:I) = ACHAR(INPARMS(I))
    END DO

    CALL IGEBR2D(ICTXT,'A',' ',20,1,INPARMS,20,0,0)
    DO I = 1, 20
        PREC(I:I) = ACHAR(INPARMS(I))
    END DO

    CALL IGEBR2D(ICTXT,'A',' ',1,1,IPART,1,0,0)
    CALL IGEBR2D(ICTXT,'A',' ',1,1,ITMAX,1,0,0)
    CALL IGEBR2D(ICTXT,'A',' ',1,1,ISTOPC,1,0,0)
    CALL IGEBR2D(ICTXT,'A',' ',1,1,ITRACE,1,0,0)

END IF

END SUBROUTINE GET_PARMS
SUBROUTINE PR_USAGE(IOUT)
    INTEGER IOUT
    WRITE(IOUT, *) ' Number of parameters is incorrect!'
    WRITE(IOUT, *) ' Use: hb_sample mtrx_file methd prec [ptype &
        &itmax istopc itrace]'
    WRITE(IOUT, *) ' Where:'
    WRITE(IOUT, *) '     mtrx_file      is stored in HB format'
    WRITE(IOUT, *) '     methd         may be: CGSTAB CGS TFQMR'
    WRITE(IOUT, *) '     prec           may be: ILU DIAGSC NONE'
    WRITE(IOUT, *) '     ptype        Partition strategy default 0'
    WRITE(IOUT, *) '                   >0: CYCLIC(ptype) '
    WRITE(IOUT, *) '                   0: BLOCK partition '
    WRITE(IOUT, *) '                   -1: Random partition '
    WRITE(IOUT, *) '     itmax          Max iterations [500] '

```

```

        WRITE(IOUT, *) '      istopc      Stopping criterion [1]      '
        WRITE(IOUT, *) '      itrace      0 (no tracing, default) or '
        WRITE(IOUT, *) '      >= 0 do tracing every ITRACE'
        WRITE(IOUT, *) '      iterations '
END SUBROUTINE PR_USAGE
END PROGRAM HB_SAMPLE

```

## Sample PARTS Subroutine

This section shows sample *parts* programs.

### PART\_BLOCK (Block Data Distribution)

```

C
C User defined function corresponding to an HPF BLOCK partition
C
      SUBROUTINE PART_BLOCK(GLOBAL_INDX,N,NP,PV,NV)

      IMPLICIT NONE

      INTEGER, INTENT(IN)  :: GLOBAL_INDX, N, NP
      INTEGER, INTENT(OUT) :: NV
      INTEGER, INTENT(OUT) :: PV(*)
      INTEGER              :: DIM_BLOCK
      REAL(8), PARAMETER  :: PC=0.0D0
      REAL(8)             :: DDIFF
      INTEGER              :: IB1, IB2, IPV

      DIM_BLOCK = (N + NP - 1)/NP
      NV = 1
      PV(NV) = (GLOBAL_INDX - 1) / DIM_BLOCK

      IPV = PV(1)
      IB1 = IPV * DIM_BLOCK + 1
      IB2 = (IPV+1) * DIM_BLOCK

      DDIFF = DBLE(ABS(GLOBAL_INDX-IB1))/DBLE(DIM_BLOCK)
      IF (DDIFF < PC/2) THEN
C
C Overlap at the beginning of a block, with the previous proc
C
        IF (IPV>0) THEN
          NV = NV + 1
          PV(NV) = IPV - 1
        ENDIF
      ENDIF

      DDIFF = DBLE(ABS(GLOBAL_INDX-IB2))/DBLE(DIM_BLOCK)
      IF (DDIFF < PC/2) THEN
C
C Overlap at the end of a block, with the next proc
C
        IF (IPV<(NP-1)) THEN
          NV = NV + 1
          PV(NV) = IPV + 1
        ENDIF
      ENDIF

      RETURN
      END

```

### PARTBCYC (Block-Cyclic Data Distribution)

```

@process free(f90)
MODULE PARTBCYC
  PUBLIC PART_BCYC, SET_NB
  PRIVATE
  INTEGER, SAVE :: BLOCK_SIZE

```

```

CONTAINS
!
! User defined subroutine corresponding to an HPF CYCLIC(NB)
! data distribution
!
SUBROUTINE PART_BCYC(GLOBAL_INDX,N,NP,PV,NV)

    IMPLICIT NONE

    INTEGER, INTENT(IN) :: GLOBAL_INDX, N, NP
    INTEGER, INTENT(OUT) :: NV
    INTEGER, INTENT(OUT) :: PV(*)

    NV = 1
    PV(NV) = MOD((((GLOBAL_INDX+BLOCK_SIZE-1)/BLOCK_SIZE)-1),NP)
    RETURN
END SUBROUTINE PART_BCYC

SUBROUTINE SET_NB(NB, RROOT, CROOT, ICTXT)
    INTEGER :: RROOT, CROOT, ICTXT
    INTEGER :: N, MER, MEC, NPR, NPC

    CALL BLACS_GRIDINFO(ICTXT,NPR,NPC,MER,MEC)

    IF (.NOT.((RROOT>=0).AND.(RROOT<NPR).AND.&
    & (CROOT>=0).AND.(CROOT<NPC))) THEN
        WRITE(0,*) 'Fatal error in SET_NB: invalid ROOT ',&
        & 'coordinates '
        CALL BLACS_ABORT(ICTXT,-1)
        RETURN
    ENDIF

    IF ((MER==RROOT).AND.(MEC==CROOT)) THEN
        IF (NB < 1) THEN
            WRITE(0,*) 'Fatal error in SET_NB: invalid NB'
            CALL BLACS_ABORT(ICTXT,-1)
            RETURN
        ENDIF
        CALL IGEBS2D(ICTXT,'A',' ',1,1,NB,1)
    ELSE
        CALL IGEBS2D(ICTXT,'A',' ',1,1,NB,1,RROOT,CROOT)
    ENDIF
    BLOCK_SIZE = NB

    RETURN
END SUBROUTINE SET_NB
END MODULE PARTBCYC

```

## **PARTRAND (Random Data Distribution)**

```

@process free(f90) init(f90ptr)
!
! Purpose:
! Provide a set of subroutines to define a data distribution based on
! a random number generator.
! This partition does *not* generally give good performance; it may be
! useful as a model to implement a graph partitioning based
! distribution; to do this you need to alter the BUILD_RNDPART
! subroutine to make it call your favorite graph partition subroutine
! instead of the random number generator.
!
! Subroutines:
!
! BUILD_RNDPART(A,NPARTS): This subroutine will be called by the root
! process to build define the data distribution mapping.
! Input parameters:
! TYPE(D_SPMAT) :: A The input matrix. The coefficients are

```

```

!                                ignored; only the structure is used.
!      INTEGER      :: NPARTS  How many parts we are requiring to the
!                                partition utility
!
!
! DISTR_RNDPART(RROOT,CROOT,ICTXT): This subroutine will be called by
! all processes to distribute the information computed by the root
! process, to be used subsequently.
!
!
! PART_RAND : The subroutine to be passed to PESSL sparse library;
!             uses information prepared by the previous two subroutines.
!
MODULE PARTRAND
  PUBLIC PART_RAND, BUILD_RNDPART, DISTR_RNDPART
  PRIVATE
  INTEGER, POINTER, SAVE :: RAND_VECT(:)

CONTAINS

  SUBROUTINE PART_RAND(GLOBAL_INDX,N,NP,PV,NV)

    INTEGER, INTENT(IN)  :: GLOBAL_INDX, N, NP
    INTEGER, INTENT(OUT) :: NV
    INTEGER, INTENT(OUT) :: PV(*)

    IF (.NOT.ASSOCIATED(RAND_VECT)) THEN
      WRITE(0,*) 'Fatal error in PART_RAND: vector RAND_VECT ',&
        & 'not initialized'
      RETURN
    ENDIF
    IF ((GLOBAL_INDX<1).OR.(GLOBAL_INDX > SIZE(RAND_VECT))) THEN
      WRITE(0,*) 'Fatal error in PART_RAND: index GLOBAL_INDX ',&
        & 'outside RAND_VECT bounds'
      RETURN
    ENDIF
    NV = 1
    PV(NV) = RAND_VECT(GLOBAL_INDX)
    RETURN
  END SUBROUTINE PART_RAND

  SUBROUTINE DISTR_RNDPART(RROOT, CROOT, ICTXT)
    INTEGER  :: RROOT, CROOT, ICTXT
    INTEGER  :: N, MER, MEC, NPR, NPC

    CALL BLACS_GRIDINFO(ICTXT,NPR,NPC,MER,MEC)

    IF (.NOT.((RROOT>=0).AND.(RROOT<NPR).AND.&
      & (CROOT>=0).AND.(CROOT<NPC))) THEN
      WRITE(0,*) 'Fatal error in DISTR_RNDPART: invalid ROOT ',&
        & 'coordinates '
      CALL BLACS_ABORT(ICTXT,-1)
      RETURN
    ENDIF

    IF ((MER == RROOT) .AND.(MEC == CROOT)) THEN
      IF (.NOT.ASSOCIATED(RAND_VECT)) THEN
        WRITE(0,*) 'Fatal error in DISTR_RNDPART: vector RAND_VECT ',&
          & 'not initialized'
        CALL BLACS_ABORT(ICTXT,-1)
        RETURN
      ENDIF
      N = SIZE(RAND_VECT)
      CALL IGEBS2D(ICTXT,'A11',' ',1,1,N,1)
      CALL IGEBS2D(ICTXT,'A11',' ',N,1,RAND_VECT,N)
    ELSE

```



```

        CALL IGEBR2D(ICTXT,'A11',' ',1,1,N,1,RROOT,CROOT)
        IF (ASSOCIATED(RAND_VECT)) THEN
DEALLOCATE(RAND_VECT)
        ENDIF
        ALLOCATE(RAND_VECT(N),STAT=INFO)
        IF (INFO /= 0) THEN
WRITE(0,*) 'Fatal error in DISTR_RNDPART: memory allocation ',&
        & ' failure.'
        RETURN
        ENDIF
        CALL IGEBR2D(ICTXT,'A11',' ',N,1,RAND_VECT,N,RROOT,CROOT)
        ENDIF

        RETURN

END SUBROUTINE DISTR_RNDPART

SUBROUTINE BUILD_RNDPART(A,NPARTS)
    USE F90SPARSE
    TYPE(D_SPMAT) :: A
    INTEGER :: NPARTS
    INTEGER :: N, I, IB, II
    INTEGER, PARAMETER :: NB=512
    REAL(KIND(1.D0)), PARAMETER :: SEED=12345.D0
    REAL(KIND(1.D0)) :: XV(NB)

    N = A%M

    IF (ASSOCIATED(RAND_VECT)) THEN
        DEALLOCATE(RAND_VECT)
    ENDIF

    ALLOCATE(RAND_VECT(N),STAT=INFO)

    IF (INFO /= 0) THEN
        WRITE(0,*) 'Fatal error in BUILD_RNDPART: memory allocation ',&
        & ' failure.'
        RETURN
    ENDIF

    IF (NPARTS.GT.1) THEN
        DO I=1, N, NB
            IB = MIN(N-I+1,NB)
            CALL DURAND(SEED,IB,XV)
            DO II=1, IB
                RAND_VECT(I+II-1) = MIN(NPARTS-1,INT(XV(II)*NPARTS))
            ENDDO
        ENDDO
    ELSE
        DO I=1, N
            RAND_VECT(I) = 0
        ENDDO
    ENDIF

    RETURN

END SUBROUTINE BUILD_RNDPART

END MODULE PARTRAND

```

## The READ\_MAT Subroutine

```

@PROCESS FREE(F90) INIT(F90PTR)
!
! READ_MAT subroutine reads a matrix and its right hand sides,
! all stored in a BCS format file. The B field is optional,.

```

```

!
! Character                      :: filename*20
!   On Entry: name of file to be processed.
!   On Exit : unchanged.
!
! Type(D_SPMAT)                  :: A
!   On Entry: fresh variable.
!   On Exit : will contain the global sparse matrix as follows:
!       A%AS for coefficient values
!       A%IA1 for column indices
!       A%IA2 for row pointers
!       A%M  for number of global matrix rows
!       A%K  for number of global matrix columns
!
! Integer                        :: ICTXT
!   On Entry: BLACS context.
!   On Exit : unchanged.
!
! Real(Kind(1.D0)), Pointer, Optional :: B(:, :)
!   On Entry: fresh variable.
!   On Exit: will contain right hand side(s).
!
! Integer, Optional              :: inroot
!   On Entry: Index of root processor (default: 0)
!   On Exit : unchanged.
!
! Real(Kind(1.D0)), Pointer, Optional :: indwork(:)
!   On Entry/Exit: Double Precision Work Area.
!
! Integer, Pointer, Optional      :: iniwork()
!   On Entry/Exit: Integer Work Area.
!
MODULE READ_MAT
PUBLIC READMAT
CONTAINS
  SUBROUTINE READMAT (FILENAME, A, ICTXT, B, INROOT,&
    & INDWORK, INIWORK)

    USE F90SPARSE

    ! Parameters
    IMPLICIT NONE
    REAL(KIND(1.D0)), POINTER, OPTIONAL :: B(:, :)
    INTEGER                        :: ICTXT
    TYPE(D_SPMAT)                  :: A
    CHARACTER                      :: FILENAME*(*)
    INTEGER, OPTIONAL              :: INROOT
    REAL(KIND(1.D00)), POINTER, OPTIONAL :: INDWORK(:)
    INTEGER, POINTER, OPTIONAL      :: INIWORK(:)

    ! Local Variables
    INTEGER, PARAMETER             :: INFILE = 2
    CHARACTER                      :: MXTYPE*3, KEY*8, TITLE*72,&
      & INDFMT*16, PTRFMT*16, RHSFMT*20, VALFMT*20, RHSTYP
    INTEGER                        :: INDCRD, PTRCRD, TOTCRD,&
      & VALCRD, RHSCRD, NROW, NCOL, NNZERO, NELTVL, NRHS, NRHSIX
    REAL(KIND(1.D00)), POINTER     :: AS_LOC(:), DWORK(:)
    INTEGER, POINTER               :: IA1_LOC(:), IA2_LOC(:), IWORK(:)
    INTEGER                        :: D_ALLOC, I_ALLOC, IRCODE, I,&
      & J, LIWORK, LDWORK, ROOT, NPROW, NPCOL, MYPROW, MYPCOL

    IF (PRESENT(INROOT)) THEN
      ROOT = INROOT
    ELSE
      ROOT = 0
    END IF

```

```

CALL BLACS_GRIDINFO(ICTXT, NPROW, NPCOL, MYPROW, MYPCOL)

IF (MYPROW == ROOT) THEN
  WRITE(*, *) 'Start read_matrix'

  ! Open Input File
  OPEN(INFILE, FILE=FILENAME, STATUS='OLD', ERR=901, ACTION="READ")
  READ(INFILE, FMT='(A72,A8,/,5I14,/,A3,11X,4I14,/,2A16,2A20)', &
    & END=902) TITLE, KEY, TOTCRD, PTRCRD, INDCRD, VALCRD, &
    & RHSCRD, MXTYPE, NROW, NCOL, NNZERO, NELTVL, &
    & PTRFMT, INDFMT, VALFMT, RHSFMT

  A%M = NROW
  A%N = NCOL
  A%FIDA = 'CSR'
  IF (RHSCRD > 0) READ(INFILE, FMT='(A1,13X,2I14)', &
    & END=902) RHSTYP, NRHS, NRHSIX

  IF (MXTYPE == 'RUA') THEN
    ALLOCATE(A%AS(NNZERO), A%IA1(NNZERO), A%IA2(NROW + 1), &
      & STAT = IRCODE)
    IF (IRCODE <> 0) GOTO 993
    READ(INFILE, FMT=PTRFMT, END=902) (A%IA2(I), I=1, NROW+1)
    READ(INFILE, FMT=INDFMT, END=902) (A%IA1(I), I=1, NNZERO)
    READ(INFILE, FMT=VALFMT, END=902) (A%AS(I), I=1, NNZERO)

  ELSE IF (MXTYPE == 'RSA') THEN
    ! We are generally working with non-symmetric matrices, so
    ! we de-symmetrize what we are about to read
    ALLOCATE(A%AS(2*NNZERO), A%IA1(2*NNZERO), &
      & A%IA2(NROW+1), AS_LOC(2*NNZERO), &
      & IA1_LOC(2*NNZERO), IA2_LOC(NROW+1), STAT=IRCODE)
    IF (IRCODE <> 0) GOTO 993
    READ(INFILE, FMT=PTRFMT, END=902) (IA2_LOC(I), I=1, NROW+1)
    READ(INFILE, FMT=INDFMT, END=902) (IA1_LOC(I), I=1, NNZERO)
    READ(INFILE, FMT=VALFMT, END=902) (AS_LOC(I), I=1, NNZERO)

    LDWORK = MAX(NROW + 1, 2 * NNZERO)
    IF (PRESENT(INDWORK)) THEN
      IF (SIZE(INDWORK) >= LDWORK) THEN
        DWORK => INDWORK
        D_ALLOC = 0
      ELSE
        ALLOCATE(DWORK(LDWORK), STAT = IRCODE)
        D_ALLOC = 1
      END IF
    ELSE
      ALLOCATE(DWORK(LDWORK), STAT = IRCODE)
      D_ALLOC = 1
    END IF
    IF (IRCODE <> 0) GOTO 993

    LIWORK = NROW + 1
    IF (PRESENT(INIWORK)) THEN
      IF (SIZE(INIWORK) >= LIWORK) THEN
        IWORK => INIWORK
        I_ALLOC = 0
      ELSE
        ALLOCATE(IWORK(LIWORK), STAT = IRCODE)
        I_ALLOC = 1
      END IF
    ELSE
      ALLOCATE(IWORK(LIWORK), STAT = IRCODE)
      I_ALLOC = 1
    END IF
    IF (IRCODE <> 0) GOTO 993

```

```

! After this call NNZERO contains the actual value for
! desymetrized matrix
CALL DESYM(NROW, AS_LOC, IA1_LOC, IA2_LOC, A%AS, A%IA1,&
& A%IA2, IWORK, DWORK, NNZERO, 1)

DEALLOCATE(AS_LOC, IA1_LOC, IA2_LOC)
IF (D_ALLOC == 1) DEALLOCATE(DWORK)
IF (I_ALLOC == 1) DEALLOCATE(IWORK)
ELSE
WRITE(0,*) 'READ_MATRIX: matrix type not yet supported'
CALL BLACS_ABORT(ICTXT, 1)
END IF

! Read Right Hand Sides
IF (PRESENT(B) .AND. (NRHS > 0)) THEN
WRITE(0,*) 'Reading RHS'
IF (RHSTYP == 'F') THEN
ALLOCATE(B(NROW, NRHS), STAT = IRCODE)
IF (IRCODE <> 0) GOTO 993
READ(INFILE,FMT=RHSFMT,END=902) ((B(I,J), I=1,NROW),J=1,NRHS)
ELSE !(RHSTYP <> 'F')
WRITE(0,*) 'READ_MATRIX: unsupported RHS type'
END IF
END IF

CLOSE(INFILE)
WRITE(*,*) 'End READ_MATRIX'
END IF

RETURN

! Open failed
901 WRITE(0,*) 'READ_MATRIX: Could not open file ',&
& INFILE,' for input'
CALL BLACS_ABORT(ICTXT, 1)

! Unexpected End of File
902 WRITE(0,*) 'READ_MATRIX: Unexpected end of file ',INFILE,&
& ' during input'
CALL BLACS_ABORT(ICTXT, 1)

! Allocation Failed
993 WRITE(0,*) 'READ_MATRIX: Memory allocation failure'
CALL BLACS_ABORT(ICTXT, 1)

END SUBROUTINE READMAT
END MODULE READ_MAT

```

## The MAT\_DIST Subroutine

```

@process free(f90) init(f90ptr)
MODULE MAT_DIST
PUBLIC MATDIST
CONTAINS
SUBROUTINE MATDIST (A_GLOB, A, PARTS, ICONTXT, DESC_A,&
& B_GLOB, B, INROOT)
!
! An utility subroutine to distribute a matrix among processors
! according to a user defined data distribution, using PESSL
! sparse matrix subroutines.
!
! Type(D_SPMAT) :: A_GLOB
! On Entry: this contains the global sparse matrix as follows:
! A%FIDA == 'CSR'
! A%AS for coefficient values
! A%IA1 for column indices

```

```

!      A%IA2  for row pointers
!      A%M    for number of global matrix rows
!      A%K    for number of global matrix columns
!      On Exit : undefined, with unassociated pointers.
!
!      Type(D_SPMAT)                                :: A
!      On Entry: fresh variable.
!      On Exit : this will contain the local sparse matrix.
!
!      INTERFACE PARTS
!      !      ....user passed subroutine.....
!      SUBROUTINE PARTS(GLOBAL_INDX,N,NP,PV,NV)
!      IMPLICIT NONE
!      INTEGER, INTENT(IN)  :: GLOBAL_INDX, N, NP
!      INTEGER, INTENT(OUT) :: NV
!      INTEGER, INTENT(OUT) :: PV(*)
!
!      END SUBROUTINE PARTS
!      END INTERFACE
!      On Entry: subroutine providing user defined data distribution.
!      For each GLOBAL_INDX the subroutine should return
!      the list PV of all processes owning the row with
!      that index; the list will contain NV entries.
!      Usually NV=1; if NV >1 then we have an overlap in the data
!      distribution.
!
!      Integer                                :: ICONTXT
!      On Entry: BLACS context.
!      On Exit : unchanged.
!
!      Type (DESC_TYPE)                        :: DESC_A
!      On Entry: fresh variable.
!      On Exit : the updated array descriptor
!
!      Real(Kind(1.D0)), Pointer, Optional    :: B_GLOB(:)
!      On Entry: this contains right hand side.
!      On Exit :
!
!      Real(Kind(1.D0)), Pointer, Optional    :: B(:)
!      On Entry: fresh variable.
!      On Exit : this will contain the local right hand side.
!
!      Integer, Optional                      :: inroot
!      On Entry: specifies processor holding A_GLOB. Default: 0
!      On Exit : unchanged.
!
!
!
!      Use F90SPARSE
!
!      Implicit None
!
! Parameters
!      Type(D_SPMAT)                        :: A_GLOB
!      Real(Kind(1.D0)), Pointer            :: B_GLOB(:)
!      Integer                              :: ICONTXT
!      Type(D_SPMAT)                        :: A
!      Real(Kind(1.D0)), Pointer            :: B(:)
!      Type (DESC_TYPE)                     :: DESC_A
!      INTEGER, OPTIONAL                     :: INROOT
!      INTERFACE PARTS
!      !      ....user passed subroutine.....
!      SUBROUTINE PARTS(GLOBAL_INDX,N,NP,PV,NV)
!      IMPLICIT NONE
!      INTEGER, INTENT(IN)  :: GLOBAL_INDX, N, NP
!      INTEGER, INTENT(OUT) :: NV
!      INTEGER, INTENT(OUT) :: PV(*)

```

```

        END SUBROUTINE PARTS
END INTERFACE

! Local variables
Integer          :: NPROW, NPCOL, MYPROW, MYPCOL
Integer          :: IRCODE, LENGTH_ROW, I_COUNT, J_COUNT,&
    & K_COUNT, BLOCKDIM, ROOT, LIWORK, NROW, NCOL, NNZERO, NRHS,&
    & I,J,K, LL, INFO
Integer, Pointer  :: IWORK(:)
CHARACTER        :: AFMT*5, atyp*5
Type(D_SPMAT)    :: BLCK

! Executable statements

IF (PRESENT(INROOT)) THEN
    ROOT = INROOT
ELSE
    ROOT = 0
END IF

CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYPROW, MYPCOL)

IF (MYPROW == ROOT) THEN
    ! Extract information from A_GLOB
    IF (A_GLOB%FIDA /= 'CSR') THEN
        WRITE(0,*) 'Unsupported input matrix format'
        CALL BLACS_ABORT(ICONTXT,-1)
    ENDIF
    NROW = A_GLOB%M
    NCOL = A_GLOB%N
    IF (NROW /= NCOL) THEN
        WRITE(0,*) 'A rectangular matrix ? ',NROW,NCOL
        CALL BLACS_ABORT(ICONTXT,-1)
    ENDIF
    NNZERO = Size(A_GLOB%AS)
    NRHS = 1
    ! Broadcast informations to other processors
    CALL IGEBS2D(ICONTXT, 'A', ' ', 1, 1, NROW, 1)
    CALL IGEBS2D(ICONTXT, 'A', ' ', 1, 1, NCOL, 1)
    CALL IGEBS2D(ICONTXT, 'A', ' ', 1, 1, NNZERO, 1)
    CALL IGEBS2D(ICONTXT, 'A', ' ', 1, 1, NRHS, 1)
ELSE ! (MYPROW <> root)
    ! Receive informations
    CALL IGEBS2D(ICONTXT, 'A', ' ', 1, 1, NROW, 1, ROOT, 0)
    CALL IGEBS2D(ICONTXT, 'A', ' ', 1, 1, NCOL, 1, ROOT, 0)
    CALL IGEBS2D(ICONTXT, 'A', ' ', 1, 1, NNZERO, 1, ROOT, 0)
    CALL IGEBS2D(ICONTXT, 'A', ' ', 1, 1, NRHS, 1, ROOT, 0)
END IF

! Allocate integer work area
LIWORK = MAX(NPROW, NROW + NCOL)
ALLOCATE(IWORK(LIWORK), STAT = IRCODE)
IF (IRCODE <> 0) THEN
    WRITE(0,*) 'MATDIST Allocation failed'
    RETURN
ENDIF

IF (MYPROW == ROOT) THEN
    WRITE (*, FMT = *) 'Start matdist'
ENDIF

```

```

CALL PADALL(NROW,PARTS,DESC_A,ICONTXT)
CALL PSPALL(A,DESC_A,NNZ=NNZERO/NPROW)
CALL PGEALL(B,DESC_A)

! Prepare the local
ALLOCATE(BLCK%AS(NCOL),BLCK%IA1(NCOL),BLCK%IA2(2),STAT=IRCODE)
IF (IRCODE /= 0) THEN
    WRITE(0,*) 'Error on allocating BLCK'
    CALL BLACS_ABORT(ICONTXT,-1)
    STOP
ENDIF

BLCK%M = 1
BLCK%N = NCOL
BLCK%FIDA = 'CSR'
Do I_COUNT = 1, NROW
    CALL PARTS(I_COUNT,NROW,NPROW,IWORK, LENGTH_ROW)
    ! Here processors are counted 1..NPROW
    DO J_COUNT = 1, LENGTH_ROW
        K_COUNT = IWORK(J_COUNT)
        IF (MYPROW == ROOT) THEN
            BLCK%IA2(1) = 1
            BLCK%IA2(2) = 1
            DO J = A_GLOB%IA2(I_COUNT), A_GLOB%IA2(I_COUNT+1)-1
                BLCK%AS(BLCK%IA2(2)) = A_GLOB%AS(J)
                BLCK%IA1(BLCK%IA2(2)) = A_GLOB%IA1(J)
                BLCK%IA2(2) = BLCK%IA2(2) + 1
            ENDDO
            LL = BLCK%IA2(2) - 1
            IF (K_COUNT == MYPROW) THEN
                BLCK%INFOA(1) = LL
                BLCK%INFOA(2) = LL
                BLCK%INFOA(3) = 2
                BLCK%INFOA(4) = 1
                BLCK%INFOA(5) = 1
                BLCK%INFOA(6) = 1
                CALL PSPINS(A,I_COUNT,1,BLCK,DESC_A)
                CALL PGEINS(B,B_GLOB(I_COUNT:I_COUNT),DESC_A,I_COUNT)
            ELSE
                CALL IGESD2D(ICONTXT,1,1,LL,1,K_COUNT,0)
                CALL IGESD2D(ICONTXT,LL,1,BLCK%IA1,LL,K_COUNT,0)
                CALL DGESD2D(ICONTXT,LL,1,BLCK%AS,LL,K_COUNT,0)
                CALL DGESD2D(ICONTXT,1,1,B_GLOB(I_COUNT),1,K_COUNT,0)
                CALL IGERV2D(ICONTXT,1,1,LL,1,K_COUNT,0)
            ENDIF
        ELSE IF (MYPROW /= ROOT) THEN
            IF (K_COUNT == MYPROW) THEN
                CALL IGERV2D(ICONTXT,1,1,LL,1,ROOT,0)
                BLCK%IA2(1) = 1
                BLCK%IA2(2) = LL+1
                CALL IGERV2D(ICONTXT,LL,1,BLCK%IA1,LL,ROOT,0)
                CALL DGERV2D(ICONTXT,LL,1,BLCK%AS,LL,ROOT,0)
                CALL DGERV2D(ICONTXT,1,1,B_GLOB(I_COUNT),1,ROOT,0)
                CALL IGESD2D(ICONTXT,1,1,LL,1,ROOT,0)
                CALL PSPINS(A,I_COUNT,1,BLCK,DESC_A)
                CALL PGEINS(B,B_GLOB(I_COUNT:I_COUNT),DESC_A,I_COUNT)
            ENDIF
        ENDIF
    END DO
END DO

! Default storage format for sparse matrix; we do not
! expect duplicated entries.
AFMT = 'DEF'
ATYP = 'GEN'
CALL PSPASB(A,DESC_A,INFO=INFO,MTYPE=ATYP,STOR=AFMT,DUPFLAG=2)

```

```

CALL PGEASB(B,DESC_A)

DEALLOCATE(BLCK%AS,BLCK%IA1,BLCK%IA2,IWORK)

IF (MYPROW == root) Write (*, FMT = *) 'End matdist'

RETURN

END SUBROUTINE MATDIST
END MODULE MAT_DIST

```

## The DESYM Subroutine

```

SUBROUTINE DESYM(NROW,A,JA,IA,AS,JAS,IAS,IAW,WORK,NNZERO,
+ VALUE)
IMPLICIT NONE
C .. Scalar Arguments ..
C INTEGER NROW,NNZERO,VALUE,INDEX
C .. Array Arguments ..
DOUBLE PRECISION A(*),AS(*),WORK(*)
INTEGER IA(*),IAS(*),JAS(*),JA(*),IAW(*)
C .. Local Scalars ..
INTEGER I,IAW1,IAW2,IAWT,J,JPT,K,KPT,LDIM,COUNT,JS,BUFI
C REAL*8 BUF
C ..

DO I=1,NROW
IAW(I)=0
END DO
C ....Compute element belonging to each row in output matrix....
DO I=1,NROW
DO J=IA(I),IA(I+1)-1
IAW(I)=IAW(I)+1
IF (JA(J).NE.I) IAW(JA(J))=IAW(JA(J))+1
END DO
END DO

IAS(1)=1
DO I=1,NROW
IAS(I+1)=IAS(I)+IAW(I)
IAW(I)=0
END DO

C
C ....Computing values array AS and column array indices JAS....
C
DO I=1,NROW
DO J=IA(I),IA(I+1)-1
IF (VALUE.NE.0) THEN
AS(IAS(I)+IAW(I))=A(J)
ENDIF
JAS(IAS(I)+IAW(I))=JA(J)
IAW(I)=IAW(I)+1
IF (I.NE.JA(J)) THEN
IF (VALUE.NE.0) THEN
AS(IAS(JA(J))+IAW(JA(J)))=A(J)
ENDIF
NNZERO=NNZERO+1
JAS(IAS(JA(J))+IAW(JA(J)))=I
IAW(JA(J))=IAW(JA(J))+1
END IF
END DO
END DO

C .....Sorting output arrays by column index.....

```



```

C      ....the IAS index not must be modified.....
C
      DO I=1,NROW
        CALL ISORTX(JAS(IAS(I)),1,IAS(I+1)-IAS(I),IAW)
        INDEX=IAS(I)-1
        IF (VALUE.NE.0) THEN
          DO J=1,IAS(I+1)-IAS(I)
            WORK(J)=AS(IAW(J)+INDEX)
          END DO
          DO J=1,IAS(I+1)-IAS(I)
            AS(J+INDEX)=WORK(J)
          END DO
        ENDIF
      END DO
C      ....column indices are already sorted by ISORTX...
      ENDDO
      RETURN

      END

```

---

## Sample Makefiles and Run Script for AIX

This section contains makefiles and run scripts that you can use with the sample thermal diffusion and sparse linear algebraic equations programs. It is divided into the following sections:

- “Makefiles and Run Script for Use with SMP Libraries on AIX”
- “Makefiles and Run Script for Use with GM Libraries on AIX” on page 973

### Makefiles and Run Script for Use with SMP Libraries on AIX

This section contains the following makefiles and run script for the SMP libraries:

- “32-Bit Makefile for Use with SMP Libraries on AIX”
- “64-Bit Makefile for Use with SMP Libraries on AIX” on page 970
- “Run Script for Use with SMP Libraries on AIX” on page 972

#### 32-Bit Makefile for Use with SMP Libraries on AIX

```

# Makefile to build the diffusion program, sparse solver and utility routines
#

# add rule for making mod files
.SUFFIXES: .mod

#
# Compilers and such
#

CC=mpcc_r
FORT=mpxlf_r
LINK=mpxlf_r

LDFLAGS = -lesslsmpl -lpesslsmpl -lblacssmpl $(LIB)
FCOPT = -O3 -C -qsource -qxref -qattr $(INCLUDE)

# default for include and lib directories

INCLUDE=
LIB=

## DISTRIBUTED DATA samples

# OBJs list objects module used in the diffusion program
#
OBJs = main.o scalempl.o param.o diffusion.o fourier.o

BASEOBJs = broadcast.o create.o delete.o init.o scatter_gather.o \
          pdata.o northsouth.o eastwest.o index.o

```

```

UTILOBSJ = $(BASEOBSJ) cdata.o utilities.o

UTILLIB = libputils.a

LIBOBSJ = $(UTILLIB)(broadcast.o) $(UTILLIB)(create.o) \
          $(UTILLIB)(delete.o) $(UTILLIB)(init.o)      \
          $(UTILLIB)(scatter_gather.o) $(UTILLIB)(pdata.o) \
          $(UTILLIB)(northsouth.o) $(UTILLIB)(eastwest.o) \
          $(UTILLIB)(index.o) $(UTILLIB)(cdata.o)      \
          $(UTILLIB)(utilities.o)

distribute: diffusion pdgexmp image simple

#
# Rule for building diffusion program
diffusion: $(OBSJ)
$(LINK) -o diffusion $(OBSJ) $(LDFLAGS)

pdgexmp: pdgexmp.o $(UTILLIB)
$(LINK) -o pdgexmp pdgexmp.o -L . -lputils $(LDFLAGS)

image: image.o $(UTILLIB)
$(LINK) -o image image.o -L . -lputils $(LDFLAGS)

simple: simple.o $(UTILLIB)
$(LINK) -o simple simple.o -L . -lputils $(LDFLAGS)

#rule to create the library

$(UTILLIB): $(LIBOBSJ)
ar rv $(UTILLIB) $%

# rules to create library objects

init.o: init.f pdata.o
xlf_r -c $(FFLAGS) init.f

exchange.o: init.f pdata.o
xlf_r -c $(FFLAGS) exchange.f

cdata.o: cdata.f pdata.o
xlf_r -c $(FFLAGS) cdata.f

create.o: create.f pdata.o
xlf_r -c $(FFLAGS) create.f

broadcast.o: broadcast.f pdata.o
xlf_r -c $(FFLAGS) broadcast.f

delete.o: delete.f pdata.o
xlf_r -c $(FFLAGS) delete.f

scatter_gather.o: scatter_gather.f pdata.o
xlf_r -c $(FFLAGS) scatter_gather.f

utilities.o: utilities.f $(BASEOBSJ) cdata.o
xlf_r -c $(FFLAGS) utilities.f

eastwest.o: eastwest.f pdata.o
xlf_r -c $(FFLAGS) eastwest.f

northsouth.o: northsouth.f pdata.o
xlf_r -c $(FFLAGS) northsouth.f

$(UTILLIB)(broadcast.o): broadcast.o
ar rv $(UTILLIB) $%

$(UTILLIB)(create.o): create.o
ar rv $(UTILLIB) $%

$(UTILLIB)(delete.o): delete.o
ar rv $(UTILLIB) $%

```

```

$(UTILLIB)(init.o): init.o
ar rv $(UTILLIB) %

$(UTILLIB)(scatter_gather.o): scatter_gather.o
ar rv $(UTILLIB) %

$(UTILLIB)(pdata.o): pdata.o
ar rv $(UTILLIB) %

$(UTILLIB)(northsouth.o): northsouth.o
ar rv $(UTILLIB) %

$(UTILLIB)(eastwest.o): eastwest.o
ar rv $(UTILLIB) %

$(UTILLIB)(index.o): index.o
ar rv $(UTILLIB) %

$(UTILLIB)(cdata.o): cdata.o
ar rv $(UTILLIB) %

$(UTILLIB)(putilities.o): utilities.o
ar rv $(UTILLIB) %

#
# List of object module dependencies for distributed data sample.
main.o: main.f scalemod.o param.o diffusion.o fourier.o
diffusion.o: diffusion.f scalemod.o param.o
fourier.o: fourier.f diffusion.o scalemod.o param.o
scalemod.o: scalemod.f param.o
param.o: param.f
pdgexmp.o: pdgexmp.f $(UTILLIB)
simple.o: simple.f $(UTILLIB)
image.o: image.f $(UTILLIB)

## SPARSE MATRIX samples

# HBOBJS and PARTOBS list objects module used in the SPARSE programs.
#
HBOBJS=read_mat.o mat_dist.o desym.o
PARTOBS= part_block.o partbcyc.o partrand.o

sparse: hb_sample pde90 pde77

pde90: pde90.o part_block.o
$(LINK) $(LDFLAGS) pde90.o part_block.o -o pde90

pde77: pde77.o part_block.o
$(LINK) $(LDFLAGS) pde77.o part_block.o -o pde77

hb_sample: $(HBOBJS) hb_sample.o $(PARTOBS)
$(LINK) $(LDFLAGS) hb_sample.o -o hb_sample \
$(HBOBJS) $(PARTOBS)

#
# List of object module dependencies for sparse matrix sample.
$(HBOBJS) hb_sample.o: read_mat.mod mat_dist.mod part_bcyc.mod partrand.mod
part_bcyc.mod: partbcyc.o

#
# Rule to clean executable and program
cleanall:
rm -f *.lst *.o *.mod diffusion core image pdgexmp simple hb_sample pde90 pde77 libutils.a

#
clean:
/bin/rm -f *.o *.mod *.lst

#
# definitions for compiles
.f.mod:
$(FORT) $(INCLUDE) $(FCOPT) -c $<

```

```
.c.o:
$(CC) $(INCLUDE) $(CCOPT) -c $<
.f.o:
$(FORT) $(INCLUDE) $(FCOPT) -c $<
```

## 64-Bit Makefile for Use with SMP Libraries on AIX

```
# Makefile to build the diffusion program, sparse solver and utility routines
#

# add rule for making mod files
.SUFFIXES: .mod

#
# Compilers and such
#

CC=mpcc_r
FORT=mpxlf_r
LINK=mpxlf_r

LDFLAGS = -lesslsm -lpesslsm -lblacssmp $(LIB)
FCOPT = -q64 -O3 -C -qsource -qxref -qattr $(INCLUDE)

# default for include and lib directories

INCLUDE=-I/usr/lpp/pessl.rte.common/include/64
LIB=

## DISTRIBUTED DATA samples

# OBJS list objects module used in the diffusion program
#
OBJS = main.o scaled.o param.o diffusion.o fourier.o

BASEOBJS = broadcast.o create.o delete.o init.o scatter_gather.o \
          pdata.o northsouth.o eastwest.o index.o

UTIOBJS = $(BASEOBJS) cdata.o putilities.o

UTILLIB = libutils.a

LIBOBJS = $(UTILLIB)(broadcast.o) $(UTILLIB)(create.o) \
          $(UTILLIB)(delete.o) $(UTILLIB)(init.o) \
          $(UTILLIB)(scatter_gather.o) $(UTILLIB)(pdata.o) \
          $(UTILLIB)(northsouth.o) $(UTILLIB)(eastwest.o) \
          $(UTILLIB)(index.o) $(UTILLIB)(cdata.o) \
          $(UTILLIB)(utilities.o)

distribute: diffusion pdgexmp image simple

#
# Rule for building diffusion program
diffusion: $(OBJS)
$(LINK) -q64 -o diffusion $(OBJS) $(LDFLAGS)

pdgexmp: pdgexmp.o $(UTILLIB)
$(LINK) -q64 -o pdgexmp pdgexmp.o -L . -lutils $(LDFLAGS)

image: image.o $(UTILLIB)
$(LINK) -q64 -o image image.o -L . -lutils $(LDFLAGS)

simple: simple.o $(UTILLIB)
$(LINK) -q64 -o simple simple.o -L . -lutils $(LDFLAGS)

#rule to create the library

$(UTILLIB): $(LIBOBJS)
ar rv $(UTILLIB) %

# rules to create library objects

init.o: init.f pdata.o
```

```

xlf_r -q64 -c $(FFLAGS) init.f

exchange.o: init.f pdata.o
xlf_r -q64 -c $(FFLAGS) exchange.f

cdata.o: cdata.f pdata.o
xlf_r -q64 -c $(FFLAGS) cdata.f

create.o: create.f pdata.o
xlf_r -q64 -c $(FFLAGS) create.f

broadcast.o: broadcast.f pdata.o
xlf_r -q64 -c $(FFLAGS) broadcast.f

delete.o: delete.f pdata.o
xlf_r -q64 -c $(FFLAGS) delete.f

scatter_gather.o: scatter_gather.f pdata.o
xlf_r -q64 -c $(FFLAGS) scatter_gather.f

utilities.o: utilities.f $(BASEOBSJS) cdata.o
xlf_r -q64 -c $(FFLAGS) utilities.f

eastwest.o: eastwest.f pdata.o
xlf_r -q64 -c $(FFLAGS) eastwest.f

northsouth.o: northsouth.f pdata.o
xlf_r -q64 -c $(FFLAGS) northsouth.f

$(UTILLIB)(broadcast.o): broadcast.o
ar -X64 rv $(UTILLIB) $%

$(UTILLIB)(create.o): create.o
ar -X64 rv $(UTILLIB) $%

$(UTILLIB)(delete.o): delete.o
ar -X64 rv $(UTILLIB) $%

$(UTILLIB)(init.o): init.o
ar -X64 rv $(UTILLIB) $%

$(UTILLIB)(scatter_gather.o): scatter_gather.o
ar -X64 rv $(UTILLIB) $%

$(UTILLIB)(pdata.o): pdata.o
ar -X64 rv $(UTILLIB) $%

$(UTILLIB)(northsouth.o): northsouth.o
ar -X64 rv $(UTILLIB) $%

$(UTILLIB)(eastwest.o): eastwest.o
ar -X64 rv $(UTILLIB) $%

$(UTILLIB)(index.o): index.o
ar -X64 rv $(UTILLIB) $%

$(UTILLIB)(cdata.o): cdata.o
ar -X64 rv $(UTILLIB) $%

$(UTILLIB)(utilities.o): utilities.o
ar -X64 rv $(UTILLIB) $%

#
# List of object module dependencies for distributed data sample.
main.o: main.f scalemod.o param.o diffusion.o fourier.o
diffusion.o: diffusion.f scalemod.o param.o
fourier.o: fourier.f diffusion.o scalemod.o param.o
scalemod.o: scalemod.f param.o
param.o: param.f
pdgexmp.o: pdgexmp.f $(UTILLIB)
simple.o: simple.f $(UTILLIB)
image.o: image.f $(UTILLIB)

```

```

## SPARSE MATRIX samples

# HBOBJS and PARTOBJJS list objects module used in the SPARSE programs.
#
HBOBJS=read_mat.o mat_dist.o desym.o
PARTOBJJS= part_block.o partbcyc.o partrand.o

sparse: hb_sample pde90 pde77

pde90: pde90.o part_block.o
$(LINK) $(LDFLAGS) -q64 pde90.o part_block.o -o pde90

pde77: pde77.o part_block.o
$(LINK) $(LDFLAGS) -q64 pde77.o part_block.o -o pde77

hb_sample: $(HBOBJS) hb_sample.o $(PARTOBJJS)
$(LINK) $(LDFLAGS) -q64 hb_sample.o -o hb_sample \
$(HBOBJS) $(PARTOBJJS)

#
# List of object module dependencies for sparse matrix sample.
$(HBOBJS) hb_sample.o: read_mat.mod mat_dist.mod part_bcyc.mod partrand.mod
part_bcyc.mod: partbcyc.o

#
# Rule to clean executable and program
cleanall:
rm -f *.lst *.o *.mod diffusion core image pdgexmp simple hb_sample pde90 pde77 libutils.a

#
clean:
/bin/rm -f *.o *.mod *.lst

#
# definitions for compiles
.f.mod:
$(FORT) $(INCLUDE) $(FCOPT) -c $<
.c.o:
$(CC) $(INCLUDE) $(CCOPT) -c $<
.f.o:
$(FORT) $(INCLUDE) $(FCOPT) -c $<

```

## Run Script for Use with SMP Libraries on AIX

```

#!/bin/ksh
#####
# LICENSED MATERIALS - PROPERTY OF IBM #
# "RESTRICTED MATERIALS OF IBM" #
# #
# 5765-422 #
# 5765-C41 #
# 5765-F84 #
# (C) COPYRIGHT IBM CORP. 1993, 2005. ALL RIGHTS RESERVED. #
# #
# U.S. GOVERNMENT USERS RESTRICTED RIGHTS - USE, DUPLICATION #
# OR DISCLOSURE RESTRICTED BY GSA ADP SCHEDULE CONTRACT WITH #
# IBM CORP. #
#####
# #
# File: run.script #
# Script file to execute a sample program. #
# The first argument is the name of the sample to run #
# The remaining arguments are any needed to by the sample #
# #
#####
# #
# USING THE FORTRAN EXAMPLE #
# Copy the files from /usr/lpp/pess1.rte.common/example/fortran #
# to a directory that is part of a shared file system (e.g. NFS #
# mounted). You can not run the program from a private file system. #
# #
# You must have the same userid on the home node and each remote node. #
# #
# You must have remote execution authority on all the nodes. #

```

```

#                                                     #
# Invoke 'make'.                                     #
#                                                     #
# Invoke 'run.script'.                               #
#                                                     #
#####

# Set the number of tasks to be 4.
export MP_PROCS=4

# Set the program to run in ip.
export MP_EUILIB=ip

# Do not use LoadLeveler to allocate nodes
export MP_RESD=no

# Use a temporary host list file.
export MP_HOSTFILE=.run.script.host.list

# Use low information output level.
export MP_INFOLEVEL=1

# Type of parallel application
export MP_PGMMODEL=smpd

# Standard output is not node ordered.
export MP_STDOUTMODE=unordered

# Determines whether incoming packets trigger interrupts (performance related)
export MP_CSS_INTERRUPT=yes

# Time between when POE checks communication of remote nodes (0 turns feature off)
export MP_PULSE=0

# Label standard io by task number.
export MP_LABELIO=yes

# If temp host file exists, delete it
if [ -s $MP_HOSTFILE ]
then
    rm $MP_HOSTFILE
fi

# Create temporary host list file
(( idx = 0 ))
while (( $idx < $MP_PROCS ))
do
    hostname >> $MP_HOSTFILE
    (( idx = idx + 1 ))
done

# Run executable
poe $*

# Remove temporary host list file
rm $MP_HOSTFILE

```

## Makefiles and Run Script for Use with GM Libraries on AIX

This section contains the following makefile and run script for the GM libraries:

- “32-Bit Makefile for Use with GM Libraries on AIX”
- “64-Bit Makefile for Use with GM Libraries on AIX” on page 976
- “Run Script for Use with GM Libraries on AIX” on page 979

### 32-Bit Makefile for Use with GM Libraries on AIX

```

#####
# LICENSED MATERIALS - PROPERTY OF IBM                #
# "RESTRICTED MATERIALS OF IBM"                       #
#                                                       #
# 5765-422                                             #

```

```

# 5765-C41 #
# 5765-F84 #
# (C) COPYRIGHT IBM CORP. 1995, 2005. ALL RIGHTS RESERVED. #
# #
# U.S. GOVERNMENT USERS RESTRICTED RIGHTS - USE, DUPLICATION #
# OR DISCLOSURE RESTRICTED BY GSA ADP SCHEDULE CONTRACT WITH #
# IBM CORP. #
#####
# #
# Makefile to build the diffusion program, sparse solver and utility #
# routines. #
#####

# add rule for making mod files
.SUFFIXES: .mod

#
# Compilers and such
#

CC=mpicc
FORT=mpif77
LINK=mpif77

LDFLAGS = -lessl -lpesslgm -lblacsgm $(LIB)
FCOPT = -O3 -C -qsource -qxref -qattr $(INCLUDE)

# default for include and lib directories

INCLUDE=
LIB=

## DISTRIBUTED DATA samples

# OBJs list objects module used in the diffusion program
#
OBJs = main.o scalem.o param.o diffusion.o fourier.o

BASEOBJs = broadcast.o create.o delete.o init.o scatter_gather.o \
          pdata.o northsouth.o eastwest.o index.o

UTILOBJs = $(BASEOBJs) cdata.o putilities.o

UTILLIB = libputils.a

LIBOBJs = $(UTILLIB)(broadcast.o) $(UTILLIB)(create.o) \
          $(UTILLIB)(delete.o) $(UTILLIB)(init.o) \
          $(UTILLIB)(scatter_gather.o) $(UTILLIB)(pdata.o) \
          $(UTILLIB)(northsouth.o) $(UTILLIB)(eastwest.o) \
          $(UTILLIB)(index.o) $(UTILLIB)(cdata.o) \
          $(UTILLIB)(putilities.o)

distribute: diffusion pdgexmp image simple

#
# Rule for building diffusion program
diffusion: $(OBJs)
$(LINK) -o diffusion $(OBJs) $(LDFLAGS)

pdgexmp: pdgexmp.o $(UTILLIB)
$(LINK) -o pdgexmp pdgexmp.o -L . -lputils $(LDFLAGS)

image: image.o $(UTILLIB)
$(LINK) -o image image.o -L . -lputils $(LDFLAGS)

simple: simple.o $(UTILLIB)
$(LINK) -o simple simple.o -L . -lputils $(LDFLAGS)

#rule to create the library

$(UTILLIB): $(LIBOBJs)
ar rv $(UTILLIB) %

```



```

# rules to create library objects

init.o: init.f pdata.o
xlf -c $(FFLAGS) init.f

exchange.o: init.f pdata.o
xlf -c $(FFLAGS) exchange.f

cdata.o: cdata.f pdata.o
xlf -c $(FFLAGS) cdata.f

create.o: create.f pdata.o
xlf -c $(FFLAGS) create.f

broadcast.o: broadcast.f pdata.o
xlf -c $(FFLAGS) broadcast.f

delete.o: delete.f pdata.o
xlf -c $(FFLAGS) delete.f

scatter_gather.o: scatter_gather.f pdata.o
xlf -c $(FFLAGS) scatter_gather.f

utilities.o: utilities.f $(BASEOBSJS) cdata.o
xlf -c $(FFLAGS) utilities.f

eastwest.o: eastwest.f pdata.o
xlf -c $(FFLAGS) eastwest.f

northsouth.o: northsouth.f pdata.o
xlf -c $(FFLAGS) northsouth.f

$(UTILLIB)(broadcast.o): broadcast.o
ar rv $(UTILLIB) %

$(UTILLIB)(create.o): create.o
ar rv $(UTILLIB) %

$(UTILLIB)(delete.o): delete.o
ar rv $(UTILLIB) %

$(UTILLIB)(init.o): init.o
ar rv $(UTILLIB) %

$(UTILLIB)(scatter_gather.o): scatter_gather.o
ar rv $(UTILLIB) %

$(UTILLIB)(pdata.o): pdata.o
ar rv $(UTILLIB) %

$(UTILLIB)(northsouth.o): northsouth.o
ar rv $(UTILLIB) %

$(UTILLIB)(eastwest.o): eastwest.o
ar rv $(UTILLIB) %

$(UTILLIB)(index.o): index.o
ar rv $(UTILLIB) %

$(UTILLIB)(cdata.o): cdata.o
ar rv $(UTILLIB) %

$(UTILLIB)(utilities.o): utilities.o
ar rv $(UTILLIB) %

#
# List of object module dependencies for distributed data sample.
main.o: main.f scalemod.o param.o diffusion.o fourier.o
diffusion.o: diffusion.f scalemod.o param.o
fourier.o: fourier.f diffusion.o scalemod.o param.o
scalemod.o: scalemod.f param.o
param.o: param.f
pdgexmp.o: pdgexmp.f $(UTILLIB)
simple.o: simple.f $(UTILLIB)

```

```

image.o: image.f $(UTILLIB)

## SPARSE MATRIX samples

# HBOBJS and PARTOBS list objects module used in the SPARSE programs.
#
HBOBJS=read_mat.o mat_dist.o desym.o
PARTOBS= part_block.o partbcyc.o partrand.o

sparse: hb_sample pde90 pde77

pde90: pde90.o part_block.o
$(LINK) $(LDFLAGS) pde90.o part_block.o -o pde90

pde77: pde77.o part_block.o
$(LINK) $(LDFLAGS) pde77.o part_block.o -o pde77

hb_sample: $(HBOBJS) hb_sample.o $(PARTOBS)
$(LINK) $(LDFLAGS) hb_sample.o -o hb_sample \
$(HBOBJS) $(PARTOBS)

#
# List of object module dependencies for sparse matrix sample.
$(HBOBJS) hb_sample.o: read_mat.mod mat_dist.mod part_bcyc.mod partrand.mod
part_bcyc.mod: partbcyc.o

#
# Rule to clean executable and program
cleanall:
rm -f *.lst *.o *.mod diffusion core image pdgexmp simple hb_sample pde90 pde77 libutils.a

#
clean:
/bin/rm -f *.o *.mod *.lst

#
# definitions for compiles
.f.mod:
$(FORT) $(INCLUDE) $(FCOPT) -c $<
.c.o:
$(CC) $(INCLUDE) $(CCOPT) -c $<
.f.o:
$(FORT) $(INCLUDE) $(FCOPT) -c $<

```

## 64-Bit Makefile for Use with GM Libraries on AIX

```

#####
# LICENSED MATERIALS - PROPERTY OF IBM #
# "RESTRICTED MATERIALS OF IBM" #
# #
# 5765-F84 #
# (C) COPYRIGHT IBM CORP. 2005. ALL RIGHTS RESERVED. #
# #
# U.S. GOVERNMENT USERS RESTRICTED RIGHTS - USE, DUPLICATION #
# OR DISCLOSURE RESTRICTED BY GSA ADP SCHEDULE CONTRACT WITH #
# IBM CORP. #
#####
# #
# Makefile to build the diffusion program, sparse solver and utility #
# routines. #
# #
#####

# add rule for making mod files
.SUFFIXES: .mod

#
# Compilers and such
#

CC=mpicc
FORT=mpif77
LINK=mpif77

```

```

LDFLAGS = -lessl -lpesslgm -lblacsgm $(LIB)
FCOPT = -q64 -O3 -C -qsource -qxref -qattr $(INCLUDE)

# default for include and lib directories

INCLUDE=-I/usr/lpp/pessl.rte.common/include/64
LIB=

## DISTRIBUTED DATA samples

# OBJs list objects module used in the diffusion program
#
OBJs = main.o scalemod.o param.o diffusion.o fourier.o

BASEOBJs = broadcast.o create.o delete.o init.o scatter_gather.o \
          pdata.o northsouth.o eastwest.o index.o

UTIOBJs = $(BASEOBJs) cdata.o putilities.o

UTILLIB = libputils.a

LIBOBJs = $(UTILLIB)(broadcast.o) $(UTILLIB)(create.o) \
          $(UTILLIB)(delete.o) $(UTILLIB)(init.o) \
          $(UTILLIB)(scatter_gather.o) $(UTILLIB)(pdata.o) \
          $(UTILLIB)(northsouth.o) $(UTILLIB)(eastwest.o) \
          $(UTILLIB)(index.o) $(UTILLIB)(cdata.o) \
          $(UTILLIB)(putilities.o)

distribute: diffusion pdgexmp image simple

#
# Rule for building diffusion program
diffusion: $(OBJs)
$(LINK) -q64 -o diffusion $(OBJs) $(LDFLAGS)

pdgexmp: pdgexmp.o $(UTILLIB)
$(LINK) -q64 -o pdgexmp pdgexmp.o -L . -lputils $(LDFLAGS)

image: image.o $(UTILLIB)
$(LINK) -q64 -o image image.o -L . -lputils $(LDFLAGS)

simple: simple.o $(UTILLIB)
$(LINK) -q64 -o simple simple.o -L . -lputils $(LDFLAGS)

#rule to create the library

$(UTILLIB): $(LIBOBJs)
ar rv $(UTILLIB) %

# rules to create library objects

init.o: init.f pdata.o
xlf -q64 -c $(FFLAGS) init.f

exchange.o: init.f pdata.o
xlf -q64 -c $(FFLAGS) exchange.f

cdata.o: cdata.f pdata.o
xlf -q64 -c $(FFLAGS) cdata.f

create.o: create.f pdata.o
xlf -q64 -c $(FFLAGS) create.f

broadcast.o: broadcast.f pdata.o
xlf -q64 -c $(FFLAGS) broadcast.f

delete.o: delete.f pdata.o
xlf -q64 -c $(FFLAGS) delete.f

scatter_gather.o: scatter_gather.f pdata.o
xlf -q64 -c $(FFLAGS) scatter_gather.f

```

```

utilities.o: utilities.f $(BASEOBSJS) cdata.o
xlf -q64 -c $(FFLAGS) utilities.f

eastwest.o: eastwest.f pdata.o
xlf -q64 -c $(FFLAGS) eastwest.f

northsouth.o: northsouth.f pdata.o
xlf -q64 -c $(FFLAGS) northsouth.f

$(UTILLIB)(broadcast.o): broadcast.o
ar -X64 rv $(UTILLIB) $%

$(UTILLIB)(create.o): create.o
ar -X64 rv $(UTILLIB) $%

$(UTILLIB)(delete.o): delete.o
ar -X64 rv $(UTILLIB) $%

$(UTILLIB)(init.o): init.o
ar -X64 rv $(UTILLIB) $%

$(UTILLIB)(scatter_gather.o): scatter_gather.o
ar -X64 rv $(UTILLIB) $%

$(UTILLIB)(pdata.o): pdata.o
ar -X64 rv $(UTILLIB) $%

$(UTILLIB)(northsouth.o): northsouth.o
ar -X64 rv $(UTILLIB) $%

$(UTILLIB)(eastwest.o): eastwest.o
ar -X64 rv $(UTILLIB) $%

$(UTILLIB)(index.o): index.o
ar -X64 rv $(UTILLIB) $%

$(UTILLIB)(cdata.o): cdata.o
ar -X64 rv $(UTILLIB) $%

$(UTILLIB)(utilities.o): utilities.o
ar -X64 rv $(UTILLIB) $%

#
# List of object module dependencies for distributed data sample.
main.o: main.f scalemod.o param.o diffusion.o fourier.o
diffusion.o: diffusion.f scalemod.o param.o
fourier.o: fourier.f diffusion.o scalemod.o param.o
scalemod.o: scalemod.f param.o
param.o: param.f
pdgexmp.o: pdgexmp.f $(UTILLIB)
simple.o: simple.f $(UTILLIB)
image.o: image.f $(UTILLIB)

## SPARSE MATRIX samples

# HBOBJS and PARTOBSJS list objects module used in the SPARSE programs.
#
HBOBJS=read_mat.o mat_dist.o desym.o
PARTOBSJS= part_block.o partbcyc.o partrand.o

sparse: hb_sample pde90 pde77

pde90: pde90.o part_block.o
$(LINK) $(LDFFLAGS) -q64 pde90.o part_block.o -o pde90

pde77: pde77.o part_block.o
$(LINK) $(LDFFLAGS) -q64 pde77.o part_block.o -o pde77

hb_sample: $(HBOBJS) hb_sample.o $(PARTOBSJS)
$(LINK) $(LDFFLAGS) -q64 hb_sample.o -o hb_sample \
$(HBOBJS) $(PARTOBSJS)

#

```

```

# List of object module dependencies for sparse matrix sample.
$(HBOBJS) hb_sample.o: read_mat.mod mat_dist.mod part_bccy.mod partrand.mod
part_bccy.mod: partbccy.o

#
# Rule to clean executable and program
cleanall:
rm -f *.lst *.o *.mod diffusion core image pdgexmp simple hb_sample pde90 pde77 libutils.a

#
clean:
/bin/rm -f *.o *.mod *.lst

#
# definitions for compiles
.f.mod:
$(FORT) $(INCLUDE) $(FCOPT) -c $<
.c.o:
$(CC) $(INCLUDE) $(CCOPT) -c $<
.f.o:
$(FORT) $(INCLUDE) $(FCOPT) -c $<

```

## Run Script for Use with GM Libraries on AIX

```

#!/bin/ksh
#####
# LICENSED MATERIALS - PROPERTY OF IBM #
# "RESTRICTED MATERIALS OF IBM" #
# #
# 5765-422 #
# 5765-C41 #
# 5765-F84 #
# (C) COPYRIGHT IBM CORP. 2005. ALL RIGHTS RESERVED. #
# #
# U.S. GOVERNMENT USERS RESTRICTED RIGHTS - USE, DUPLICATION #
# OR DISCLOSURE RESTRICTED BY GSA ADP SCHEDULE CONTRACT WITH #
# IBM CORP. #
#####
#
# File: run.scriptgm #
# Script file to execute a sample program with MPICH-GM. #
# The first argument is the name of the sample to run #
# The remaining arguments are any needed to by the sample #
# #
#####
#
# USING THE FORTRAN EXAMPLE #
# Copy the files from /usr/lpp/pess1.rte.common/example/fortran #
# to a directory that is part of a shared file system (e.g. NFS #
# mounted). You can not run the program from a private file system. #
# #
# You must have the same userid on the home node and each remote node. #
# #
# You must have remote execution authority on all the nodes. #
# #
# Invoke 'make -f Makefilegm'. #
# #
# Invoke 'run.scriptgm'. #
# #
#####

# Set the number of tasks to be 4.
export MP_PROCS=4

# Set the program to run in MPICH-GM environment.
export MP_EUILIB=gm

# Do not use LoadLeveler to allocate nodes
export MP_RESD=no

# Use a temporary host list file.
export MP_HOSTFILE=.run.script.host.list

# Use low information output level.

```

```

export MP_INFOLEVEL=1

# Type of parallel application
export MP_PGMMODEL=spmd

# Standard output is not node ordered.
export MP_STDOUTMODE=unordered

# Determines whether incoming packets trigger interrupts (performance related)
export MP_CSS_INTERRUPT=yes

# Time between when POE checks communication of remote nodes (0 turns feature off)
export MP_PULSE=0

# Label standard io by task number.
export MP_LABELIO=yes

# If temp host file exists, delete it
if [ -s $MP_HOSTFILE ]
then
    rm $MP_HOSTFILE
fi

# Create temporary host list file
(( idx = 0 ))
while (( $idx < $MP_PROCS ))
do
    hostname >> $MP_HOSTFILE
    (( idx = idx + 1 ))
done

# Run executable
mpirun -np $MP_PROCS -machinefile $MP_HOSTFILE $*

# Remove temporary host list file
rm $MP_HOSTFILE

```

---

## Sample Makefiles and Run Script for Linux

You can use the following makefile and run script with the sample thermal diffusion and sparse linear algebraic equations programs.

### 32-Bit Makefile for Linux

```

# Makefile to build the diffusion program, sparse solver and utility routines
#
# To compile the sample programs, MPICH-GM scripts must be in your
# execution path or you must specify the full path where the MPICH-GM
# scripts are located.
#
# To run the sample programs, xlf_r must be in your execution path or you must
# specify the full path where xlf_r is located.
#PATH:=$(PATH):/opt/ibmcmp/xlf/version.release/bin
#
#
# add rule for making mod files
.SUFFIXES: .mod

#
# Compilers and such
#

FORT=mpif77 -fc=xlf_r
LINK=mpif77 -fc=xlf_r
XLF=xlf_r -c

LDFLAGS = $(libs) $(LIB)
FCOPT = -O3 -C -qsource -qxref -qattr $(INCLUDE)

# default for include and lib directories

```

```

INCLUDE =
LIB=
libs= -lpslgm -lblacsgm -lessl

## DISTRIBUTED DATA samples

# OBJs lists the objects used in the diffusion program
#
OBJs = main.o scaled.o param.o diffusion.o fourier.o

BASEOBJs = broadcast.o create.o delete.o init.o scatter_gather.o \
          pdata.o northsouth.o eastwest.o index.o

UTILOBJs = $(BASEOBJs) cdata.o utilities.o

UTILLIB = libputils.a

LIBOBJs = $(UTILLIB)(broadcast.o) $(UTILLIB)(create.o) \
          $(UTILLIB)(delete.o) $(UTILLIB)(init.o) \
          $(UTILLIB)(scatter_gather.o) $(UTILLIB)(pdata.o) \
          $(UTILLIB)(northsouth.o) $(UTILLIB)(eastwest.o) \
          $(UTILLIB)(index.o) $(UTILLIB)(cdata.o) \
          $(UTILLIB)(utilities.o)

distribute: diffusion pdgexmp image simple

#
# Rule for building diffusion program
diffusion: $(OBJs)
$(LINK) -o diffusion $(OBJs) $(libs) -lessl $(LIB)

pdgexmp: pdgexmp.o $(UTILLIB)
$(LINK) -o pdgexmp pdgexmp.o -L . -lputils $(LDFLAGS)

image: image.o $(UTILLIB)
$(LINK) -o image image.o -L . -lputils $(LDFLAGS)

simple: simple.o $(UTILLIB)
$(LINK) -o simple simple.o -L . -lputils $(LDFLAGS)

#rule to create the library

$(UTILLIB): $(LIBOBJs)
ar rv $(UTILLIB) $%

# rules to create library objects

init.o: init.f pdata.o
$(XLF) $(FFLAGS) init.f

exchange.o: init.f pdata.o
$(XLF) $(FFLAGS) exchange.f

cdata.o: cdata.f pdata.o
$(XLF) $(FFLAGS) cdata.f

create.o: create.f pdata.o
$(XLF) $(FFLAGS) create.f

broadcast.o: broadcast.f pdata.o
$(XLF) $(FFLAGS) broadcast.f

delete.o: delete.f pdata.o
$(XLF) $(FFLAGS) delete.f

scatter_gather.o: scatter_gather.f pdata.o
$(XLF) $(FFLAGS) scatter_gather.f

utilities.o: utilities.f $(BASEOBJs) cdata.o
$(XLF) $(FFLAGS) utilities.f

eastwest.o: eastwest.f pdata.o
$(XLF) $(FFLAGS) eastwest.f

```

```

northsouth.o: northsouth.f pdata.o
$(XLF) $(FFLAGS) northsouth.f

$(UTILLIB)(broadcast.o): broadcast.o
ar rv $(UTILLIB) $$

$(UTILLIB)(create.o): create.o
ar rv $(UTILLIB) $$

$(UTILLIB)(delete.o): delete.o
ar rv $(UTILLIB) $$

$(UTILLIB)(init.o): init.o
ar rv $(UTILLIB) $$

$(UTILLIB)(scatter_gather.o): scatter_gather.o
ar rv $(UTILLIB) $$

$(UTILLIB)(pdata.o): pdata.o
ar rv $(UTILLIB) $$

$(UTILLIB)(northsouth.o): northsouth.o
ar rv $(UTILLIB) $$

$(UTILLIB)(eastwest.o): eastwest.o
ar rv $(UTILLIB) $$

$(UTILLIB)(index.o): index.o
ar rv $(UTILLIB) $$

$(UTILLIB)(cdata.o): cdata.o
ar rv $(UTILLIB) $$

$(UTILLIB)(utilities.o): utilities.o
ar rv $(UTILLIB) $$

#
# List of object dependencies for distributed data sample.
main.o: main.f scalemod.o param.o diffusion.o fourier.o
diffusion.o: diffusion.f scalemod.o param.o
fourier.o: fourier.f diffusion.o scalemod.o param.o
scalemod.o: scalemod.f param.o
param.o: param.f
pdgexmp.o: pdgexmp.f $(UTILLIB)
simple.o: simple.f $(UTILLIB)
image.o: image.f $(UTILLIB)

## SPARSE MATRIX samples

# HBOBJS and PARTOBSJS list objects used in the SPARSE programs.
#
HBOBJS=read_mat.o mat_dist.o desym.o
PARTOBSJS= part_block.o partbcyc.o partrand.o

sparse: hb_sample pde90 pde77

pde90: pde90.o part_block.o
$(LINK) $(LD_FLAGS) pde90.o part_block.o -o pde90

pde77: pde77.o part_block.o
$(LINK) $(LD_FLAGS) pde77.o part_block.o -o pde77

hb_sample: $(HBOBJS) hb_sample.o $(PARTOBSJS)
$(LINK) $(LD_FLAGS) hb_sample.o -o hb_sample \
$(HBOBJS) $(PARTOBSJS)

#
# List of object dependencies for sparse matrix sample.
$(HBOBJS) hb_sample.o: read_mat.mod mat_dist.mod part_bcyc.mod partrand.mod
part_bcyc.mod: partbcyc.o

```



```

#
# Rule to clean executable and program
cleanall:
    rm -f *.lst *.o *.mod diffusion core image pdgexmp simple hb_sample pde90 pde77 libputils.a

#
clean:
    /bin/rm -f *.o *.mod *.lst

#
# definitions for compiles
.f.mod:
$(FORT) $(INCLUDE) $(FCOPT) -c $<
.f.o:
$(FORT) $(INCLUDE) $(FCOPT) -c $<

```

## 64-Bit Makefile for Linux

```

# To compile the sample programs, MPICH-GM scripts must be in your
# execution path or you must specify the full path where the MPICH-GM
# scripts are located.
#
# To run the sample programs, xlf_r must be in your execution path or you must
# specify the full path where xlf_r is located.
#PATH=$(PATH):/opt/ibmcmp/xlf/version.release/bin
#
# add rule for making mod files
.SUFFIXES: .mod

#
# Compilers and such
#

FORT=mpif77 -fc=xlf_r -q64
LINK=mpif77 -fc=xlf_r -q64

XLF=xlf_r -q64 -c

LDFLAGS = $(libs) $(LIB)
FCOPT = -O3 -C -qsource -qxref -qattr $(INCLUDE)

# default for include and lib directories

INCLUDE= -I/opt/ibmmath/pessl/3.2/include64
LIB=
libs= -lpesslgm -lblacsgm -lessl

## DISTRIBUTED DATA samples

# OBJS lists objects used in the diffusion program
#
OBJS = main.o scalemod.o param.o diffusion.o fourier.o

BASEOBJ = broadcast.o create.o delete.o init.o scatter_gather.o \
        pdata.o northsouth.o eastwest.o index.o

UTIOBJ = $(BASEOBJ) cdata.o utilities.o

UTILLIB = libputils.a

LIBOBJ = $(UTILLIB)(broadcast.o) $(UTILLIB)(create.o) \
        $(UTILLIB)(delete.o) $(UTILLIB)(init.o) \
        $(UTILLIB)(scatter_gather.o) $(UTILLIB)(pdata.o) \
        $(UTILLIB)(northsouth.o) $(UTILLIB)(eastwest.o) \
        $(UTILLIB)(index.o) $(UTILLIB)(cdata.o) \
        $(UTILLIB)(utilities.o)

distribute: diffusion pdgexmp image simple

#
# Rule for building diffusion program
diffusion: $(OBJ)

```

```

$(LINK) -o diffusion $(OBS) $(libs) -lessl $(LIB)

pdgexp: pdgexp.o $(UTILLIB)
$(LINK) -o pdgexp pdgexp.o -L . -lputils $(LD_FLAGS)

image: image.o $(UTILLIB)
$(LINK) -o image image.o -L . -lputils $(LD_FLAGS)

simple: simple.o $(UTILLIB)
$(LINK) -o simple simple.o -L . -lputils $(LD_FLAGS)

#rule to create the library

$(UTILLIB): $(LIBOBS)
ar rv $(UTILLIB) %

# rules to create library objects

init.o: init.f pdata.o
$(XLF) $(FFLAGS) init.f

exchange.o: init.f pdata.o
$(XLF) $(FFLAGS) exchange.f

cdata.o: cdata.f pdata.o
$(XLF) $(FFLAGS) cdata.f

create.o: create.f pdata.o
$(XLF) $(FFLAGS) create.f

broadcast.o: broadcast.f pdata.o
$(XLF) $(FFLAGS) broadcast.f

delete.o: delete.f pdata.o
$(XLF) $(FFLAGS) delete.f

scatter_gather.o: scatter_gather.f pdata.o
$(XLF) $(FFLAGS) scatter_gather.f

utilities.o: utilities.f $(BASEOBS) cdata.o
$(XLF) $(FFLAGS) utilities.f

eastwest.o: eastwest.f pdata.o
$(XLF) $(FFLAGS) eastwest.f

northsouth.o: northsouth.f pdata.o
$(XLF) $(FFLAGS) northsouth.f

$(UTILLIB)(broadcast.o): broadcast.o
ar rv $(UTILLIB) %

$(UTILLIB)(create.o): create.o
ar rv $(UTILLIB) %

$(UTILLIB)(delete.o): delete.o
ar rv $(UTILLIB) %

$(UTILLIB)(init.o): init.o
ar rv $(UTILLIB) %

$(UTILLIB)(scatter_gather.o): scatter_gather.o
ar rv $(UTILLIB) %

$(UTILLIB)(pdata.o): pdata.o
ar rv $(UTILLIB) %

$(UTILLIB)(northsouth.o): northsouth.o
ar rv $(UTILLIB) %

$(UTILLIB)(eastwest.o): eastwest.o
ar rv $(UTILLIB) %

$(UTILLIB)(index.o): index.o
ar rv $(UTILLIB) %

```

```

$(UTILLIB)(cdata.o): cdata.o
ar rv $(UTILLIB) $%

$(UTILLIB)(utilities.o): utilities.o
ar rv $(UTILLIB) $%

#
# List of object dependencies for distributed data sample.
main.o: main.f scalemod.o param.o diffusion.o fourier.o
diffusion.o: diffusion.f scalemod.o param.o
fourier.o: fourier.f diffusion.o scalemod.o param.o
scalemod.o: scalemod.f param.o
param.o: param.f
pdgexmp.o: pdgexmp.f $(UTILLIB)
simple.o: simple.f $(UTILLIB)
image.o: image.f $(UTILLIB)

## SPARSE MATRIX samples

# HBOBJS and PARTOBS list objects used in the SPARSE programs.
#
HBOBJS=read_mat.o mat_dist.o desym.o
PARTOBS= part_block.o partbcyc.o partrand.o

sparse: hb_sample pde90 pde77

pde90: pde90.o part_block.o
$(LINK) $(LDFLAGS) pde90.o part_block.o -o pde90

pde77: pde77.o part_block.o
$(LINK) $(LDFLAGS) pde77.o part_block.o -o pde77

hb_sample: $(HBOBJS) hb_sample.o $(PARTOBS)
$(LINK) $(LDFLAGS) hb_sample.o -o hb_sample \
$(HBOBJS) $(PARTOBS)

#
# List of object dependencies for sparse matrix sample.
$(HBOBJS) hb_sample.o: read_mat.mod mat_dist.mod part_bcyc.mod partrand.mod
part_bcyc.mod: partbcyc.o

#
# Rule to clean executable and program
cleanall:
rm -f *.lst *.o *.mod diffusion core image pdgexmp simple hb_sample pde90 pde77 libputils.a

#
clean:
/bin/rm -f *.o *.mod *.lst

#
# definitions for compiles
.f.mod:
$(FORT) $(INCLUDE) $(FCOPT) -c $<
.f.o:
$(FORT) $(INCLUDE) $(FCOPT) -c $<

```

## Run Script for Linux

```

# USING THE FORTRAN EXAMPLE
# Copy the files from /opt/ibmmath/pessl/3.2/example/fortran to
# a directory that
# is part of a shared file system (e.g. NFS mounted). You can not run
# the program from a local file system.
#
# You must have the same userid on the home node and each remote node.
#
# You must have remote execution authority on all the nodes.
#

```

```

# Invoke 'make' and 'make sparse' to generate 32-bit Parallel ESSL Fortran
# examples. Invoke 'make -f Makefile64' and 'make -f Makefile64 sparse'
# to generate 64-bit Parallel ESSL fortran examples.
#
# Invoke 'run.script'.
#
#
# Script file to execute a sample program.
# The first argument is the name of the sample to run
# The remaining arguments are needed by the sample
#
# Set the number of tasks to be 8.
MP_PROCS=8
#
# hostlist contains the list of machines to run the sample program.
#
machine_file=hostlist
mpirun.ch_gm -np $MP_PROCS -machinefile $machine_file $*

```

---

## Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation  
Licensing  
2-31 Roppongi 3-chome, Minato-ku  
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation  
Intellectual Property Law  
2455 South Road, P386  
Poughkeepsie, NY 12601  
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

#### COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

---

## Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States or other countries or both:

AIX  
AIX 5L  
e (logo)  
IBM  
IBMLink  
POWER2  
PowerPC

pSeries  
RS/6000  
Scalable POWERparallel Systems  
SP  
System/370  
System/390

UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.

Other company, product, and service names may be the trademarks or service marks of others.

---

## Software Update Protocol

IBM has provided modifications to this software. The resulting software is provided to you on an “AS IS” basis and WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

---

## Programming Interfaces

This *IBM Parallel Engineering and Scientific Subroutine Library Guide and Reference* manual is intended to help the customer to do application programming. This manual documents General-use Programming Interface and Associated Guidance Information provided by Parallel ESSL.

General-use programming interfaces allow the customer to write programs that obtain the services of Parallel ESSL.





---

## Glossary

This glossary defines terms and abbreviations used in this publication. If you do not find the term you are looking for, refer to the index portion of this book. This glossary includes terms and definitions from:

- *IBM Dictionary of Computing*, New York: McGraw Hill (1-800-2MC-GRAW), 1994.
- *American National Standard Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 11 West 42nd Street, New York, New York 10036. Definitions are identified by the symbol (A) after the definition.
- *Information Technology Vocabulary*, developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions from published sections of these vocabularies are identified by the symbol (I) after the definition. Definitions taken from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol (T) after the definition, indicating that final agreement has not yet been reached among the participating National Bodies of SC1.

IBM is grateful to the American National Standards Institute (ANSI) for permission to reprint its definitions from the American National Standard *Vocabulary for Information Processing* (Copyright 1970 by American National Standards Institute, Incorporated), which was prepared by Subcommittee X3K5 on Terminology and Glossary of the American National Standards Committee X3. ANSI definitions are preceded by an asterisk (\*).

### A

**address.** A character or group of characters that identifies a register, a device, a particular part of storage, or some other data source or destination.

**AIX.** Abbreviation for Advanced Interactive Executive, IBM's licensed version of the UNIX® operating system.

AIX is particularly suited to support technical computing applications, including high function graphics and floating point computations.

**Amd.** Berkely Software Distribution automount daemon.

**APAR.** Authorized Program Analysis Report. A report of a problem caused by a suspected defect in a current unaltered release of a program.

**application.** The use to which a data processing system is put; for example, a computational chemistry application, a signal processing application.

**application data.** The data that is produced using an application program.

**argument.** A parameter passed between a calling program and a SUBROUTINE subprogram, a FUNCTION subprogram, or a statement function.

**array.** An ordered set of data items identified by a single name.

**array descriptor.** Contains the information required to establish the mapping between a global data structure and its corresponding process and memory location.

**array element.** A data item in an array, identified by the array name followed by a subscript indicating its position in the array.

**array name.** The name of an ordered set of data items that make up an array.

**assignment statement.** A statement that assigns a value to a variable or array element. It is made up of a variable or array element, followed by an equal sign (=), followed by an expression. The variable, array element, or expression can be character, logical, or arithmetic. When the assignment statement is processed, the expression to the right of the equal sign replaces the value of the variable or array element to the left.

### B

**bandwidth.** The total available bit rate of a digital channel.

**Basic Linear Algebra Communication Subprograms (BLACS).** A standard set of public domain subroutines that perform message passing (communications) between processes.

**Basic Linear Algebra Subprograms (BLAS).** A standard set of public domain mathematical subroutines that perform linear algebra operations.

**BLACS.** Basic Linear Algebra Communication Subprograms.

**BLAS.** Basic Linear Algebra Subprograms.

## C

**cache.** A high-speed buffer.

**character constant.** A string of one or more alphanumeric characters enclosed in apostrophes. The delimiting apostrophes are not part of the value of the constant.

**character expression.** An expression in the form of a single character constant, variable, array element, substring, function reference, or another expression enclosed in parentheses. A character expression is always of type character.

**character type.** The data type for representing strings of alphanumeric characters; in storage, one byte is used for each character.

**client.** \* (1) A function that requests services from a server, and makes them available to the user. \* (2) A term used in an environment to identify a machine that uses the resources of the network.

**cluster.** A group of processors interconnected through a high speed network that can be used for high performance computing.

**column-major order.** A sequencing method used for storing multidimensional arrays according to the subscripts of the array elements. In this method the leftmost subscript position varies most rapidly and completes a full cycle before the next subscript position to the right is incremented.

**CMI.** Centralized Management Interface provides a series of SMIT menus and dialogues used for defining and querying the SP system configuration.

**complex conjugate even data.** Complex data that has its real part even and its imaginary part odd.

**complex constant.** An ordered pair of real or integer constants separated by a comma and enclosed in parentheses. The first real constant of the pair is the real part of the complex number; the second is the imaginary part.

**complex type.** The data type for representing an approximation of the value of a complex number. A data item of this type consists of an ordered pair of real data items separated by a comma and enclosed in

parentheses. The first item represents the real part of the complex number; the second represents the imaginary part.

**constant.** An unvarying quantity. The four classes of constants specify numbers (arithmetic), truth values (logical), character data (character), and hexadecimal data.

## D

**daemon.** A process, not associated with a particular user, that performs system-wide functions such as administration and control of networks, execution of time-dependent activities, line printer spooling, etc.

**default.** An alternative value, attribute, or option that is assumed when none has been specified.

**dataless workstation.** A workstation that has local disks which may be used for **swap**, **tmp**, and **usr** file systems.

**data distribution.** The method in which global data structures are divided among processes. Three types of data distribution are: cyclic, block-cyclic, and block distribution.

**data type.** The structural characteristics, features and properties of data that may be directly specified by a programming language; for example, integers, real numbers in Fortran; arrays in APL; linked lists in LISP; character string in SNOBOL.

**decimation.** The formation of a sequence containing every n-th element of another sequence.

**dimension of an array.** One of the subscript expression positions in a subscript for an array. In Fortran, an array may have from one to seven dimensions. Graphically, the first dimension is represented by the rows, the second by the columns, and the third by the planes. Contrast with rank. See also extent of a dimension.

**direct access storage.** A storage device in which the access time is in effect independent of the location of the data. (A)

**diskless workstation.** A computer workstation with its own processor, keyboard, graphics system and monitor but no local disk system. The system relies on disk resources which are found in the network either on a dedicated server or shared over the entire network resources.

**divide-by-zero exception.** The condition recognized by a processor that results from running a program that attempts to divide by zero.

**DNS.** Domain Name Server is a hierarchical name service which maps high level machine names to IP addresses.

**double precision.** Synonym for long-precision.

**DWM.** Diskless Workstation Manager is operating-system software that initializes and maintains resources for diskless clients and diskless servers.

## E

**Ethernet.** Ethernet is the standard hardware for TCP/IP LANs in the UNIX marketplace. It is a 10 megabit per second baseband type network that uses the contention based CSMA/CD (collision detect) media access method.

**expression.** A notation that represents a value: a primary appearing alone, or combinations of primaries and operators. An expression can be arithmetic, character, logical, or relational.

**extent of a dimension.** The number of different integer values that may be represented by subscript expressions for a particular dimension in a subscript for an array.

**external function.** A function defined outside the program unit that refers to it. It may be referred to in a procedure subprogram or in the main program, but it must not refer to itself, either directly or indirectly. Contrast with statement function.

## F

**file.** A set of related records treated as a unit, for example, in stock control, a file could consist of a set of invoices.

**file server.** A centrally located computer that acts as a storehouse of data and applications for numerous users of a local area network.

**foreign host.** Any host on the network other than the local host.

**FTP.** File transfer protocol.

**function.** In Fortran, a procedure that is invoked by referring to it in an expression and that supplies a value to the expression. The value supplied is the value of the function. See also external function, intrinsic function, and statement function. Contrast with subroutine.

**function reference.** A Fortran source program reference to an intrinsic function, to an external function, or to a statement function.

## G

**general matrix.** A matrix with no assumed special properties such as symmetry. Synonym for matrix.

**global.** (1) Pertaining to that which is defined in one subdivision of a computer program and used in at least one other subdivision of the computer program. (2) Pertaining to information available to more than one program or subroutine. (3) Contrast with local.

## H

**home directory.** The directory associated with an individual user.

**host.** A computer connected to a network, and providing an access method to that network. A host provides end-user services.

## I

**integer constant.** A string of decimal digits containing no decimal point and expressing a whole number.

**integer expression.** An arithmetic expression whose values are of integer type.

**integer type.** An arithmetic data type capable of expressing the value of an integer. It can have a positive, negative, or 0 value. It must not include a decimal point.

**Internet.** The collection of worldwide networks and gateways which function as a single, cooperative virtual network.

**internet address.** A unique 32-bit address assigned to hosts connected to a TCP/IP network.

**intrinsic function.** A function, supplied by Fortran, that performs mathematical or character operations.

**IP.** Internet protocol.

## K

**kernel.** The core portion of the UNIX operating system which controls the resources of the CPU and allocates them to the users. The kernel is memory-resident, is said to run in "kernel mode" and is protected from user tampering by the hardware.

## L

**LAN.** Acronym for Local Area Network, a data network located on the user's premises in which serial transmission is used for direct data communication among data stations.

**latency.** The time interval between the instant at which an instruction control unit initiates a call for data transmission and the instant at which the actual transfer of data begins. Latency is related to the hardware characteristics of the system and to the

different layers of software that are involved in initiating the task of packing and transmitting the data.

**leading dimension.** For a two-dimensional array, an increment used to find the starting point for the matrix elements in each successive column of the array.

**local.** Pertaining to that which is defined and used only in one subdivision of a computer program. Contrast with global.

**local host.** The computer to which a user's terminal is directly connected.

**logical constant.** A constant that can have one of two values: true or false. The form of these values in Fortran is: .TRUE. and .FALSE. respectively.

**logical expression.** A logical primary alone or a combination of logical primaries and logical operators. A logical expression can have one of two values: true or false.

**logical type.** The data type for data items that can have the value true or false and upon which logical operations such as .NOT. and .OR. can be performed. See also "data type".

**long-precision.** Real type of data of length 8. Contrast with single precision and short-precision.

## M

**main program.** In Fortran, a program unit, required for running, that can call other program units but cannot be called by them.

**mainframe.** A large computer to which other computers can be connected, so that they can share facilities that the large computer provides; for example, it could be a System/370 or System/390 computing system to which personal computers are attached, so that they can upload and download programs and data.

**mask.** To use a pattern of characters to control the retention or elimination of portions of another pattern of characters. (I)

**matrix.** A rectangular array of elements, arranged in rows and columns, that may be manipulated according to the rules of matrix algebra. (A) (I)

**menu.** A display of a list of available functions for selection by the user.

**message passing.** The method of communication among processor nodes operating in parallel with distributed memory.

**MPI.** A Message Passing Interface standard.

**MPMD (Multiple Program - Multiple Data).** A parallel programming model in which different, but related, programs are run on different sets of data.

## N

**name.** In Fortran, a string of up to six alphanumeric characters, the first of which must be alphabetic. Used to identify a constant, a variable, an array, a function, a subroutine, or a common block.

**network.** An interconnected group of nodes, lines, and terminals. A network provides the ability to transmit data to and receive data from other systems and users.

**NFS.** Network file system. NFS allows different systems (UNIX or non-UNIX), different architectures, or vendors connected to the same network, to access remote files in a LAN environment as though they were local files.

**node.** In a network, the point where one or more functional units interconnect transmission lines. A computer location defined in a network.

**nodeid.** The specific symbolic name assigned to a node during network definition.

## O

**overflow exception.** A condition caused by the result of an arithmetic operation having a magnitude that exceeds the largest possible number.

## P

**parallel processing.** A multiprocessor architecture which allows processes to be allocated to tightly coupled multiple processors in a cooperative processing environment, allowing concurrent execution of tasks.

**parameter.** \* (1) A variable that is given a constant value for a specified application and that may denote the application. (2) An item in a menu for which the operator specifies a value or for which the system provides a value when the menu is interpreted. (3) A name in a procedure that is used to refer to an argument that is passed to the procedure. (4) A particular piece of information that a system or application program needs to process a request.

**pipe.** A UNIX utility allowing the output of one command to be the input of another. Represented by the | symbol. It is also referred to as filtering output.

**port.** (1) An endpoint for communication between devices, generally referring to physical connection. (2) A 16-bit number identifying a particular TCP or UDP resource within a given TCP/IP node.

**primary.** An irreducible unit of data; a single constant, variable, array element, function reference, or expression enclosed in parentheses.

**process.** \* (1) A unique, finite course of events defined by its purpose or by its effect, achieved under defined conditions. \* (2) Any operation or combination of operations on data. \* (3) A function being performed or waiting to be performed. \* (4) A program in operation. For example, a daemon is a system process that is always running on the system.

**process grid.** A way to view a parallel machine as a logical one- or two-dimensional rectangular grid of processes.

**program exception.** The condition recognized by a processor that results from running a program that improperly specifies or uses instructions, operands, or control information.

**protocol.** A set of semantic and syntactic rules that defines the behavior of functional units in achieving communication.

**PDF.** Portable Document Format.

**PTF.** Program Temporary Fix. A temporary solution or by-pass of a problem diagnosed by IBM as resulting from a defect in a current unaltered release of the program. A report of a problem caused by a suspected defect in a current unaltered release of a program.

## R

**rank.** In Fortran, the number of dimensions of an array. It is zero for scalar.

**real constant.** A string of decimal digits that expresses a real number. A real constant must contain either a decimal point or a decimal exponent and may contain both. For example, the real constant 0.36819E+2 has the value +36.819.

**real type.** An arithmetic data type, capable of approximating the value of a real number. It can have a positive, negative, or 0 value.

**remote host.** See *foreign host*.

**row-major order.** A sequencing method used for storing multidimensional arrays according to the subscripts of the array elements. In this method the rightmost subscript position varies most rapidly and completes a full cycle before the next subscript position to the left is incremented.

**RISC.** Reduced Instruction Set Computing (RISC), the technology for today's high performance personal computers and workstations, was invented in 1975.

## S

**scalar.** (1) A quantity characterized by a single number. (A) (I) (2) Contrast with vector.

**scope.** (1) The portion of a computer program within which the definition of a variable remains unchanged. (2) In Appendix A, "BLACS Quick Reference Guide," on page 891, for the broadcast topologies and global operations, scope can equal 'all', 'row', or 'column'.

**server.** (1) A function that provides services for users. A machine may run client and server processes at the same time. (2) A machine that provides resources to the network. It provides a network service, such as disk storage and file transfer, or a program that uses such a service.

**ScaLAPACK (Scalable Linear Algebra Package).** A scalable linear algebra library for distributed memory concurrent computers. The library was jointly developed by the University of Tennessee, Knoxville, Oak Ridge National Laboratory, and the University of California, Berkeley.

**shape of an array.** The extents of all the dimensions of an array listed in order. For example, the shape of a three-dimensional array that has four rows, five columns, and three planes is (4,5,3) or 4 by 5 by 3.

**shell.** The shell is the primary user interface for the UNIX operating system. It serves as command language interpreter, programming language, and allows foreground and background processing. There are three different implementations of the shell concept: Bourne, C and Korn.

**short-precision.** Real type data of length 4. Contrast with double precision and long-precision.

**single precision.** Synonym for short-precision.

**size of an array.** The number of elements in an array. This is the product of the extents of its dimensions.

**SMIT.** The System Management Interface Toolkit is a set of menu driven utilities for AIX that provides functions such as transaction login, shell script creation, automatic updates of object data base, etc.

**SMP.** Symmetric Multi-Processing.

**SPMD (Single Program - Multiple Data).** A parallel programming model in which different processors execute the same program on different sets of data.

**statement.** The basic unit of a program, that specifies an action to be performed, or the nature and characteristics of the data to be processed, or information about the program itself. Statements fall into two broad classes: executable and nonexecutable.



**statement function.** A procedure specified by a single statement that is similar in form to an arithmetic, logical, or character assignment statement. The statement must appear after the specification statements and before the first executable statement. In the remainder of the program it can be referenced as a function. A statement function may be referred to only in the program unit in which it is defined. Contrast with external function.

**statement label.** A number of from one through five decimal digits that is used to identify a statement. Statement labels can be used to transfer control, to define the range of a DO, or to refer to a FORMAT statement.

**statement number.** See "statement label".

**stride.** The increment used to step through array storage to select the vector or matrix elements from the array.

**subprogram.** A program unit that is invoked by another program unit in the same program. In Fortran, a subprogram has a FUNCTION, SUBROUTINE, or BLOCK DATA statement as its first statement.

**subscript.** (1) A symbol that is associated with the name of a set to identify a particular subset or element. (A) (2) A subscript expression or set of subscript expressions, enclosed in parentheses and used with an array name to identify a particular array element.

**subscript expression.** An integer expression in a subscript whose value and position in the subscript determine the index number for the corresponding dimension in the referenced array.

**System Administrator.** The user who is responsible for setting up, modifying, and maintaining the computing system.

## T

**tar.** Tape ARchive, is a standard UNIX data archive utility for storing data on tape media.

**TCP.** Acronym for Transmission Control Protocol, a stream communication protocol that includes error recovery and flow control.

**TCP/IP.** Acronym for Transmission Control Protocol/Internet Protocol, a suite of protocols designed to allow communication between networks regardless of the technologies implemented in each network.

**Telnet.** Terminal Emulation Protocol, a TCP/IP application protocol that allows interactive access to foreign hosts.

**thread.** A thread is the element that is scheduled, and to which resources such as execution time, locks, and queues may be assigned. There may be one or more

threads in a process, and each thread is executed by the operating system concurrently.

**thread-safe.** A subroutine which may be called from multiple threads of the same process simultaneously.

**thread-tolerant.** A library is thread-tolerant if it can be called from a single thread of a multithreaded application. However, multiple simultaneous calls to the thread-tolerant library from different threads of a single process causes unpredictable results.

**transaction.** An exchange between the user and the system. Each activity the system performs for the user is considered a transaction.

**transfer.** To send data from one place and to receive the data at another place. Synonymous with move.

**transmission.** \* The sending of data from one place for reception elsewhere.

**type declaration.** The explicit specification of the type of a constant, variable, array, or function by use of an explicit type specification statement.

## U

**UDP.** User Datagram Protocol.

**underflow exception.** A condition caused by the result of an arithmetic operation having a magnitude less than the smallest possible nonzero number.

**URL.** Uniform Resource Locator.

**user.** Anyone who requires the services of a computing system.

## V

**variable.** (1) A quantity that can assume any of a given set of values. (A) (2) A data item, identified by a name, that is not a named constant, array, or array element, and that can assume different values at different times during program processing.

**vector.** A one-dimensional ordered collection of numbers.

## W

**working directory.** A collection of files to be manipulated by an FTP operation.

**workstation.** A workstation is a single-user, high-performance microcomputer (or even a minicomputer) which has been specialized in some way, usually for graphics output. Such a machine has a screen and a keyboard, but is also capable of extensive processing of your input before it is passed to the host.

Likewise, the host's responses may be extensively processed before being passed along to your screen. A workstation may be intelligent enough to do much or all the processing itself.

## X

**X Window System.** A product developed at MIT that gives users windows into applications and processes not located only or specifically on their own console or computer system.





---

## Bibliography

This bibliography lists the publications that you may need to use with Parallel ESSL and describes how to obtain them.

---

### References

Text books and articles covering the mathematical aspects of ESSL are listed in this section, as well as several software libraries available from other companies. They are listed alphabetically as follows:

- Publications are listed by the author's name. IBM publications that include an order number, other than an *IBM Technical Report* can be ordered through the Subscription Library Services System (SLSS). The non-IBM publications listed here should be obtained through publishers, bookstores, or professional computing organizations.
- Software libraries are listed by their product name. Each reference includes the names, addresses, and phone numbers of the companies from which they can be obtained.

Each citation in the text of this book is shown as a number enclosed in square brackets. It indicates the number of the item listed in the bibliography. For example, reference [1] cites the first item listed below.

1. Agarwal, R. C.; Cooley, J. W. September 1987. "Vectorized Mixed Radix Discrete Fourier Transform Algorithms." *IEEE Proceedings*, Vol. 75-9:1283-1292.
2. Agarwal, R. C.; Gustavson, F.; Joshi, M.; Zubair, M. February 1995. "A Scalable Parallel Block Algorithm for Band Cholesky Factorization." SIAM Conference on Parallel Processing.
3. Agarwal, R. C.; Gustavson, F.; Zubair, M. May 1994. "An Efficient Parallel Algorithm for the Three-Dimensional FFT NAS Parallel Benchmark." *Proceedings of IEEE SHPPC '94*, 129-133.
4. Anderson, E.; Bai, Z.; Bischof, C.; Demmel, J.; Dongarra, J.; DuCroz, J.; Greenbaum, A.; Hammarling, S.; McKenney, A.; Ostrouchov, S.; Sorensen, D. 1995. "LAPACK User's Guide, Second Edition." SIAM, Philadelphia, Pa.
5. Anderson, E.; Bai, Z.; Bischof, C.; Demmel, J.; Dongarra, J.; DuCroz, J.; Greenbaum, A.; Hammarling, S.; McKenney, A.; Sorensen, D. May 1990. "LAPACK: A Portable Linear Algebra Library for High-Performance Computers." University of Tennessee, Technical Report CS-90-105.
6. Anderson, E.; Benzoni, A.; Dongarra, J.; Moulton, S.; Ostrouchov, S.; Tourancheau, B.; van de Geijn, R. 1991. "Basic Linear Algebra Communication Subprograms." *Sixth Distributed Memory Computing Conference Proceedings* IEEE Computer Society Press.
7. Arioli, M.; Duff, I. S.; Ruiz, M. 1992. "Stopping Criteria for Iterative Solvers," *SIAM Journal of Matrix Analysis Application*, Vol. 13, pp. 138-144.
8. Bailey, D.; Barton, J.; Lasinski, T.; Simon, H.. 1991. "NAS Parallel Benchmarks." Report RNR-91-002, Revision 2. NASA Ames Research Center, Moffett Field, CA.
9. Barrett, R.; Berry, M.; Chan, T. F.; Demmel, J.; Donato, J.; Dongarra, J.; Eijkhout, V.; Pozo, R.; Romine, C.; van der Vorst, H. 1994. "Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods," SIAM. Philadelphia, PA.
10. Blackford, L. S.; Choi, J.; Cleary, A.; D'Azevedo, E.; Demmel, J.; Dhillon, I.; Dongarra, J.; Hammarling, S.; Henry, G.; Petitet, A.; Stanley, K.; Walker, D.; Whaley, R. C. 1997. "ScaLAPACK Users' Guide," SIAM. Philadelphia, PA.
11. Brainerd, W. S.; Goldberg, C. H.; Adams, J. C. 1990. *Programmer's Guide to Fortran 90* Intertext Publications and McGraw-Hill Book Company, New York, N.Y.
12. Cerioni, F.; Colajanni, M.; Filippone, S.; Maiolatesi, S. 1996. "A Proposal for Parallel Sparse BLAS," in *Proceedings of PARA '96*. Edited by J. Wasniewski, J. Dongarra, K. Madsen, and D. Olesen. Springer-Verlag *Lecture Notes in Computer Science*, No. 1184, pp. 166-175.
13. Choi, J.; Demmel, J.; Dhillon, I.; Dongarra, J. J.; Ostrouchov, L. S.; Petitet, A.; Walker, D.; Whaley, R. C.; Stanley, K. March 1995. "Installation Guide for ScaLAPACK,"

- LAPACK Working Note 93 University of Tennessee, Technical Report CS-95-280.
14. Choi, J.; Dongarra, J. J.; Ostrouchov, L. S.; Petitet, A.; Whaley, R. C.; Demmel, J.; Dhillon, I.; Stanley, K. February 1995. "Installation Guide for ScaLAPACK," *LAPACK Working Note* University of Tennessee; University of California, Berkeley.
  15. Choi, J.; Dongarra, J. J.; Walker, D. W. 1994. "PB-BLAS: A Set of Parallel Block Basic Linear Algebra Subprograms," *Technical Report ORNL/TM-12468* Oak Ridge National Laboratory, Mathematical Sciences Section, Oak Ridge, Tennessee.
  16. Choi, J.; Dongarra, J. J.; Walker, D. W. 1994. "PB-BLAS: Reference Manual," *Technical Report ORNL/TM-12469* Oak Ridge National Laboratory, Mathematical Sciences Section, Oak Ridge, Tennessee.
  17. Choi, J.; Dongarra, J. J.; Ostrouchov, S.; Petitet, A. P.; Walker, D. W.; Whaley, R. C. September 1994. "The Design and Implementation of the ScaLAPACK LU, QR, and Cholesky Factorization Routines," *LAPACK Working Note 80* University of Tennessee, Technical Report CS-94-246.
  18. Choi, J.; Dongarra, J. J.; Ostrouchov, S.; Petitet, A. P.; Walker, D. W.; Whaley, R. C. May 1995. "A Proposal for a Set of Parallel Basic Linear Algebra Subprograms," *LAPACK Working Note 100* Soongsil University, University of Tennessee, and Oak Ridge National Laboratory.
  19. Choi, J.; Dongarra, J. J.; Walker, D. W. 1994. "SCALAPACK Reference Manual I: Parallel Factorization Routines (LU, QR, and Cholesky)," *Technical Report ORNL/TM-12471* Oak Ridge National Laboratory, Mathematical Sciences Section, Oak Ridge, Tennessee.
  20. Choi, J.; Dongarra, J. J.; Walker, D. W. 1994. "SCALAPACK II: Parallel Reduction Routines (HRD, TRD, and BRD)," *Technical Report ORNL/TM-12472* Oak Ridge National Laboratory, Mathematical Sciences Section, Oak Ridge, Tennessee.
  21. Choi, J.; Dongarra, J. J.; Walker, D. W. 1994. "SCALAPACK Reference Manual II: Parallel Reduction Routines (HRD, TRD, and BRD)," *Technical Report ORNL/TM-12473* Oak Ridge National Laboratory, Mathematical Sciences Section, Oak Ridge, Tennessee.
  22. Choi, J.; Dongarra, J. J.; Walker, D. February 1995. "The Design of a Parallel Dense Linear Algebra Software Library: Reduction to Hessenberg, Tridiagonal, and Bidiagonal Form," *LAPACK Working Note 92* University of Tennessee, Technical Report CS-95-275.
  23. Choi, J.; Dongarra, J. J.; Pozo, R.; Walker, D. 1992. "ScaLAPACK: A Scalable Linear Algebra Library for Distributed Memory Concurrent Computers." *Proceedings of the Fourth Symposium on the Frontiers of Massively Parallel Computation (FRONTIERS '92)* IEEE Computer Society Press.
  24. Choi, J.; Dongarra, J. J.; Pozo, R.; Walker, D. November 1992. "ScaLAPACK: A Scalable Linear Algebra Library for Distributed Memory Concurrent Computers," University of Tennessee, Technical Report CS-92-181.
  25. Demmel, J. W.; Dhillon, I.; Ren, H. March 1994. "On the correctness of Parallel Bisection in Floating Point," *LAPACK Working Note 70* University of Tennessee, Technical Report CS-94-228.
  26. Demmel, J. W.; Kahan, W. February 1988. "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," *LAPACK Working Note 3* Argonne National Laboratory, MCS-TM-110.
  27. Demmel, J. W.; Stanley, K. September 1994. "The Performance of Finding Eigenvalues and Eigenvectors of Dense Symmetric Matrices on Distributed Floating Point," *LAPACK Working Note 86* University of Tennessee, Technical Report CS-94-254.
  28. Dongarra, J. J.; DuCroz, J.; Hammarling, S.; Duff, I. March 1990. "A Set of Level 3 Basic Linear Algebra Subprograms." *ACM Transactions on Mathematical Software*, 16(1):1-17.
  29. Dongarra, J. J.; DuCroz, J.; Hammarling, S.; Hanson, R. J. March 1988. "An Extended Set of FORTRAN Basic Linear Algebra Subprograms." *ACM Transactions on Mathematical Software*, 14(1):1-17.
  30. Dongarra, J. J.; DuCroz, J.; Hammarling, S.; Hanson, R. J. March 1988. "Algorithm 656. An Extended Set of Basic Linear Algebra Subprograms: Model Implementation and Test Programs." *ACM Transactions on Mathematical Software*, 14(1):18-32.
  31. Dongarra, J. J.; DuCroz, J.; Hammarling, S. 1996. "A Proposal for Fortran 90 BLAS," *LAPACK Working Note* University of Tennessee, Oak Ridge National Laboratory, and Numerical Algorithms Group Ltd.

32. Dongarra, J. J.; DuCroz, J.; Hammarling, S.; Wasniewski, J.; Zemla, A. August 1995. "A Proposal for a Fortran 90 Interface for LAPACK," *LAPACK Working Note 101* University of Tennessee, Numerical Algorithms Group Ltd., Technical University of Denmark, and Polish Academy of Sciences.
33. Dongarra, J. J.; van de Geijn, R. A. 1991. "Two Dimensional Basic Linear Algebra Communication Subprogram," *LAPACK Working Note 37* University of Tennessee, Technical Report CS-91-138.
34. Dongarra, J. J.; van de Geijn, R. A.; Whaley, R. C. December 1993 and June 1994. *A User's Guide to the BLACS*, Oak Ridge National Laboratory, Oak Ridge, Tennessee.
35. Dongarra, J. J.; Walker, D. 1993. "The Design of Linear Algebra Libraries for High Performance Computers," *LAPACK Working Note 58* University of Tennessee, Technical Report CS-93-188.
36. Duff, I. S.; Marrone, M.; Radicati, G.; and Vittoli, C. September 1997. "Level 3 Basic Linear Algebra Subprograms for Sparse Matrices: a User Level Interface." *ACM Transactions on Mathematical Software*, 23(3):379-401.
37. Fernando, K.; Parlett, B. 1994. "Accurate singular values and differential qd algorithms." *Numerische Mathematik*, 67:191-229.
38. Filippone, S.; Sales, M. L. 1994. "Experiences in Numerical Software on IBM Distributed Memory Architectures." *Parallel Scientific Computing* 207-218. Edited by J. Dongarra and J. Wasniewski. Springer-Verlag, New York, Heidelberg, Berlin.
39. Golub, G. H.; Van Loan, C. F. 1996. *Matrix Computations*, John Hopkins University Press, Baltimore, Maryland.
40. Gropp, W.; Lusk, E.; Skjellum, A. 1994. *Using MPI: Portable Parallel Programming with the Message Passing Interface*. The MIT Press, Cambridge, MA; London, England. (This book is also orderable through SLSS by specifying publication number SR28-5757.)
41. Gupta, A.; Gustavson, F.; Joshi, M.; Toledo, S. June, 20, 1996. "The Design, Implementation, and Evaluation of a Banded Linear Solver for Distributed-Memory Parallel Computers." *IBM Research Report RC 20481*.
42. Gupta, A.; Gustavson, F.; Joshi, M.; Toledo, S. 1996. "The Design, Implementation, and Evaluation of a Banded Linear Solver for Distributed-Memory Parallel Computers." *Applied Parallel Computing, Industrial Problems and Optimization*. Edited by Dongarra, J.; Madsen, K.; Washniewski, J. Parallel Computing, Third International Workshop, PARA'96 Lyngby, Denmark, August 1996. Proceedings Lecture Notes in Computer Science, Springer-Verlag, 1996.
43. *High Performance Fortran Language Specification* High Performance Fortran Forum; November 10, 1994; Version 1.1.
44. Holian, B. L.; Percus, O. E.; Warnock, T. T.; Whitlock, P. A. August 1994. "Pseudorandom Number Generator for Massively Parallel Molecular-Dynamics Simulations." *Physical Review E*, 50(2).
45. Kelley, C. T. 1995. "Iterative Methods for Linear and Nonlinear Equations" *SIAM*. Philadelphia, PA.
46. Koelbel, C.; Loveman, D.; Schreiber, R.; Steele Jr., G.; Zosel, M. 1994. *The High Performance Fortran Handbook*. The MIT Press, Cambridge, MA; London, England.
47. Metcalf, M.; Reid, J. 1994. *Fortran 90 Explained*. Oxford University Press, U.S.A.
48. *Message Passing Interface Forum, MPI: A Message Passing Interface Standard, Version 1.1*. June 6, 1995. University of Tennessee, Knoxville, Tennessee. This document can be obtained at the following URL:  
  
<http://www.mcs.anl.gov/Projects/mpi/index.html>
49. Parlett, B.; Marques, O. "An implementation of the dqds Algorithm (Positive Case)," *LAPACK Working Note 155*.  
  
You can download this document from:  
  
<http://www.netlib.org/lapack/lawns/lawn155.ps>
50. Percus, O. E.; Kalos, M. H. 1989. "Random Number Generators for MIMD Parallel Processors." *Journal of Parallel and Distributed Computing*, 6:477-497.
51. Percus, O. E.; Percus, J. K. July 1988. "Long Range Correlations in Linear Congruential Generators." *Journal of Computational Physics*, 77(1).
52. Percus, O. E.; Percus, J. K. 1992. "An Expanded Set of Correlation Tests for Linear

- Congruential Random Number Generators." *Combinatorics, Probability and Computing*, 1:161-168.
53. Percus, O. E.; Percus, J. K. 1992. "Intrinsic Relations in the Structure of Linear Congruential Generators Modulo  $2^b$ ." *Statistics and Probability Letters*, 15:381-383.
54. Sun, Xian-He; Zhang, Hong; Ni, Lionel M. March 1992. "Efficient Tridiagonal Solvers on Multicomputers". *IEEE Transactions on Computers*, Vol. 41, No. 3.
55. Whaley, R. Clint. May 1994. "Basic Linear Algebra Communication Subprograms: Analysis and Implementation Across Multiple Parallel Architectures," *LAPACK Working Note 73* University of Tennessee, Technical Report CS-94-234.

## Parallel ESSL Publications

Parallel ESSL publications are all available on the Internet at the URLs listed in Table 6 on page 11.

## Related Publications

The related publications or libraries at the Web sites listed below may be useful to you when using Parallel ESSL.

Product	Web site URL
AIX	<a href="http://www.ibm.com/servers/aix">http://www.ibm.com/servers/aix</a>
Linux	For general information and documentation on Linux: <a href="http://www.tldp.org/">http://www.tldp.org/</a>  For information about the standard Linux installation procedure using the RPM Package Manager (RPM): <a href="http://www.rpm.org/">http://www.rpm.org/</a>  For information about IBM-related offerings for Linux: <a href="http://www.ibm.com/linux/">http://www.ibm.com/linux/</a>
C and C++	<a href="http://www-3.ibm.com/software/awdtools/vacpp/library/">http://www-3.ibm.com/software/awdtools/vacpp/library/</a>
XL Fortran	<a href="http://www-3.ibm.com/software/awdtools/fortran/xlfortran/library/">http://www-3.ibm.com/software/awdtools/fortran/xlfortran/library/</a>
MPICH	<a href="http://www.mcs.anl.gov/Projects/mpi/index.html">http://www.mcs.anl.gov/Projects/mpi/index.html</a>
MPICH-GM	<a href="http://www.myri.com/scs">http://www.myri.com/scs</a>  For downloading shared libraries: <a href="http://ppclinux.ncsa.uiuc.edu">http://ppclinux.ncsa.uiuc.edu</a>
Parallel Environment	<a href="http://publib.boulder.ibm.com/clresctr/windows/public/pebooks.html">http://publib.boulder.ibm.com/clresctr/windows/public/pebooks.html</a>



---

# Index

## A

- abbreviations
  - for product names x
  - in the Glossary 991
  - interpreting math and programming xii
- about this book vii
- absolute value
  - notation xii
- accuracy
  - of results 5
- acronyms
  - associated with programming values xii
  - in the Glossary 991
  - product names x
- address notation xii
- advantages of Parallel ESSL 3
- algebra 351
- ANSI definitions in Glossary 991
- APAR
  - definition of 991
- application program outline
  - sparse (Fortran 77) 90
  - sparse (Fortran 90) 89
- architecture supported by Parallel ESSL 6
- arguments
  - conventions used in the subroutine descriptions xiv
  - definition of 991
  - font for ESSL calling xi
  - list of Parallel ESSL input-argument errors 108, 122
- array
  - definition of 991
  - element, definition of 991
  - name, definition 991
- array descriptor 59, 61
  - block 31
  - definition of 991
  - for block-cyclic 27
- array descriptors, migrating to this
  - release of Parallel ESSL 99
- arrow notation, what it means xii
- assignment statement, definition of 991
- attention messages, list of Parallel ESSL 107

## B

- background books 999
- band matrix, distributing 42
- bibliography 999
- BLACS (Basic Linear Algebra Communication Subprograms)
  - BLACS\_EXIT 86
  - BLACS\_GET 80
  - BLACS\_GRIDEXIT 86
  - BLACS\_GRIDINFO 83

- BLACS (Basic Linear Algebra Communication Subprograms)
  - (continued)
  - BLACS\_GRIDINIT 81
  - BLACS\_GRIDMAP 83
  - BLACS\_PINFO 80
  - initializing in your program 80
  - quick reference 891
  - usage by Parallel ESSL 12
- BLACS\_EXIT 86
- BLACS\_GET 80
- BLACS\_GRIDEXIT 86
- BLACS\_GRIDINFO 83
- BLACS\_GRIDINIT 81
- BLACS\_GRIDMAP 83
- BLACS\_PINFO 80
- BLAS (Basic Linear Algebra Subprograms)
  - definition of 991
- block column, distributing 40
- block distributing 29
- block distribution, definition of 22
- block row, distributing 40
- block size, suggested 77
- block-cyclic distribution, definition of 22
- bold letters, usage of xi
- books
  - See publications

## C

- C (C programming language)
  - modifying procedures for using Parallel ESSL for AIX 92
  - modifying procedures for using Parallel ESSL for Linux 96
  - Parallel ESSL header file 87
- C and C++
  - publications 1002
- C libraries 8
- C++ (C++ programming language)
  - modifying procedures for using Parallel ESSL AIX 93
  - modifying procedures for using Parallel ESSL for Linux 96
  - Parallel ESSL header file 87
- cache, definition of 991
- calling sequence
  - syntax description xiii
- ceiling notation and meaning xii
- character constant, definition of 991
- character data
  - conventions xi
- character expression, definition of 991
- character type, definition of 991
- characters, special usage of xii
- choosing how many MPI tasks and computational threads to use 77
- citations
  - See references, math background

- coding your program
  - application program outline 87
  - optimal performance 77
  - restrictions for routine names 79
  - where to find more information 79
- column-major order, definition of 991
- communication errors
  - list of messages for 120
  - overview 106
- comparison of accuracy 5
- compilers, required by Parallel ESSL 7
- compilers, required by Parallel ESSL for Linux 8
- compiling your program
  - C programs 92, 96
  - C++ programs 93, 96
  - Fortran programs 92, 95
- complex conjugate even data, definition of 991
- complex conjugate notation xii
- complex constant, definition of 991
- complex data
  - conventions xi
- complex type, definition of 991
- computational areas, overview 3
- computational errors
  - differs from ESSL for AIX, how this 105
  - list of messages for 118
  - overview 105
- computational threads 77
- conjugate notation xii
- constant, definition of 991
- continuation, convention for numerical data xi
- conventions
  - for messages 107
  - mathematical and programming notations xii
  - scalar data xi
  - subroutine descriptions xiii
- cosine notation xii
- courier font usage xi
- customer service, IBM 101
- customer support, IBM 101
- cyclic distribution, definition of 22

## D

- D\_SPMAT, derived data type 59
- data
  - conventions for scalar data xi
- data distribution
  - block 22
  - block-cyclic 22
  - cyclic 22
  - definition of ix, 22
  - general tridiagonal matrix over one-dimensional process grid 44

- data distribution (*continued*)
  - matrix
    - over one-dimensional process
      - grid 40
    - over two-dimensional process
      - grid 53
  - random number generation
    - subroutine 36
  - symmetric band matrix
    - over one-dimensional process
      - grid 42
  - symmetric tridiagonal matrix
    - over one-dimensional process
      - grid 48
  - techniques 22
  - three-dimensional sequences 69
  - two-dimensional sequences 64
  - vectors
    - over one-dimensional process
      - grid 33
    - over two-dimensional process
      - grid 36
- data structures, distributing across
  - processes 21
- data type, definition of 991
- decimation, definition of 991
- definitions of terms in the Glossary 991
- derived data types
  - D\_SPMAT 59
  - DESC\_TYPE 59
- DESC\_TYPE, derived data type 59
- DESCINIT 849
- descriptions, conventions used in the
  - subroutine xiii
- descriptors, migrating to this release of
  - Parallel ESSL 99
- DESCSET 852
- determinant
  - matrix notation xii
- determining the norm for a general
  - matrix 872
- determining the norm for a real
  - symmetric, complex symmetric, or
    - complex Hermitian matrix 879
- determining the number of rows or
  - columns 868
- diagnosis procedures
  - in your program 101
  - Parallel ESSL messages, list of 107
- diagnostics
  - See* messages
- dimensions of arrays
  - definition of 991
- direct access storage
  - definition of 991
- divide-by-zero exception, definition
  - of 991
- documentation
  - See* publications
- dot product
  - notation xii
- double precision, definition of 991

## E

- eigensystem analysis and singular value
  - analysis subroutines
    - overview 675
  - PDGEBRD 762
  - PDGEHRD 753
  - PDGESVD 780
  - PDSYEVX 677
  - PDSYGST 739
  - PDSYGVX 698
  - PDSYTRD 724
  - PZGEBRD 762
  - PZGESVD 780
  - PZHEEVX 677
  - PZHEGST 739
  - PZHEGVX 698
  - PZHETRD 724
- element of a matrix notation xii
- element of a vector notation xii
- Engineering and Scientific Subroutine
  - Library x
- environment variable,
  - PESSL\_ERROR\_SYNC 77, 103
- error conditions, conventions used in the
  - subroutine descriptions xv
- error messages
  - See* messages
- errors
  - communication errors 106
  - computational errors 105
  - ESSL error messages 107
  - input-argument errors 104
  - miscellaneous errors 106
  - MPI error messages 107
  - program exceptions on the
    - workstations 104
  - resource errors 105
  - synchronization 103
  - types of errors that you can
    - encounter 103
  - where to find more information
    - on 101
- ESSL error messages 107
- ESSL, Parallel (Parallel Engineering and
  - Scientific Subroutine Library)
    - advantages of 3
    - coding your program 79
    - communication errors 106
    - computational areas, overview 3
    - computational errors 105
    - diagnosis procedures 101
    - distributing your data across
      - processes 21
    - dynamic linking versus static
      - linking 92, 95
    - eigensystem analysis and singular
      - value analysis subroutines 675
    - ESSL error messages 107
    - Fourier Transforms 797
    - functional capability 3
    - input-argument errors 104
    - installation requirements 10
    - introduction to 3
    - languages supported 5
    - Level 2 PBLAS 129
    - Level 3 PBLAS 237

- ESSL, Parallel (Parallel Engineering and
  - Scientific Subroutine Library)
    - (*continued*)
      - linear algebraic equations
        - subroutines 351
      - message conventions 107
      - messages, list of 107
      - miscellaneous errors 106
      - MPI error messages 107
      - name x
      - number of subroutines in each area 3
      - overview 3
      - overview of the subroutines 3
      - packaging characteristics 10
      - parallel processing subroutines on the
        - workstations 4
      - program exceptions on the
        - workstations 104
      - program number for 1002
      - publications overview 1002
      - random number generation
        - subroutines 837
      - reference information
        - conventions xiii
      - resource errors 105
      - running your program (linking, load,
        - and run) 91, 94
      - using error handling 101
      - utility subroutines 845
- Euclidean norm notation xii
- examples, conventions used in the
  - subroutine descriptions xv
- exponential function notation xii
- expression, definition of 991
- expressions, special usage of xii
- extent of a dimension, definition of 991
- external function, definition of 991

## F

- factoring
  - general matrix 355, 370, 435
  - general tridiagonal matrix 518, 532
  - positive definite complex Hermitian
    - matrix 449
  - positive definite real symmetric
    - matrix 449
  - positive definite symmetric band
    - matrix 486, 498
  - positive definite symmetric tridiagonal
    - matrix 565, 579
- FFT-packed storage mode 65, 70
- floor notation and meaning xii
- fonts used in this book xi
- Fortran
  - languages required by Parallel
    - ESSL 7
  - languages required by Parallel ESSL
    - for Linux 8
  - modifying procedures for using
    - Parallel ESSL for AIX 92
  - modifying procedures for using
    - Parallel ESSL for Linux 95
- Fortran 90
  - overview 16
  - sparse linear algebraic equation
    - subroutines 16

Fortran 90 sample program 895  
 Fourier transform  
   overview 797  
   sequences, distributing data 64  
   three dimensions  
     complex 817  
     complex-to-real 831  
     real-to-complex 825  
   two dimensions  
     complex 800  
     complex-to-real 812  
     real-to-complex 807  
 Fourier transform subroutines  
   PDCFT2 800  
   PDCFT3 817  
   PDCRFT2 812  
   PDCRFT3 831  
   PDRCF2 807  
   PDRCF3 825  
   PSCFT2 800  
   PSCFT3 817  
   PSCRFT2 812  
   PSCRFT3 831  
   PSRCFT2 807  
   PSRCFT3 825  
 Frobenius norm notation xii  
 full block matrix, distributing 40  
 function  
   definition of 991  
 function reference, definition of 991  
 functional capability of the Parallel ESSL  
   subroutines 3

## G

General Information manual, ESSL 1002  
 general matrix 351  
 general matrix, definition of 991  
 general tridiagonal matrix,  
   distributing 44  
 generation of random numbers 837  
 global data structures, definition of 21  
 Glossary 991  
 greek letters notation xii  
 guide information 1  
 guidelines for handling problems  
   *See* diagnosis procedures

## H

hardware  
   required for Parallel ESSL 6  
 header file, Parallel ESSL 87  
 how to use this book vii, viii  
 Hypertext Markup Language, required  
   products 11

## I

IBM publications  
   *See* publications  
 ICEIL 855  
 identity matrix notation xii  
 ILCM 856  
 INDYG2L 857  
 INDYG2P 859

INDXL2G 861  
 infinity norm notation xii  
 infinity notation xii  
 INFOG1L 863  
 INFOG2L 865  
 informational messages, for Parallel  
   ESSL 107  
 input arguments, conventions used in the  
   subroutine descriptions xiv  
 input-argument errors  
   differs from ESSL for AIX, how  
     this 104  
   list of messages for 108, 122  
   overview 104  
 int notation and meaning xii  
 integer constant, definition of 991  
 integer data  
   conventions xi  
 integer expression, definition of 991  
 integer type, definition of 991  
 intrinsic function, definition of 991  
 introduction to Parallel ESSL 3  
 inverse  
   matrix notation xii  
 IPESSL  
   IPESSL 847  
 ISO definitions in Glossary 991  
 italic font usage xi

## L

languages supported by Parallel ESSL 5  
 leading dimension for matrices  
   definition of 991  
 letters, fonts of xi  
 Level 2 PBLAS  
   overview 129  
 Level 2 PBLAS subroutines  
   PDGEMV 131  
   PDGER 168  
   PDSYMV 154  
   PDSYR 186  
   PDSYR2 197  
   PDTRMV 212  
   PDTRSV 224  
   PZDTRSV 224  
   PZGEMV 131  
   PZGERC 168  
   PZGERU 168  
   PZHEMV 154  
   PZHER 186  
   PZHER2 197  
   PZTRMV 212  
 Level 3 PBLAS  
   overview 237  
 Level 3 PBLAS subroutines  
   PDGEMM 239  
   PDSYMM 256  
   PDSYR2K 316  
   PDSYRK 301  
   PDTRAN 336  
   PDTRMM 276  
   PDTRSM 288  
   PZGEMM 239  
   PZHEMM 256  
   PZHER2K 316  
   PZHERK 301

Level 3 PBLAS subroutines (*continued*)  
   PZSYMM 256  
   PZSYR2K 316  
   PZSYRK 301  
   PZTRANC 336  
   PZTRANU 336  
   PZTRMM 276  
   PZTRSM 288  
 level of Parallel ESSL, getting 847  
 library  
   overview 3  
 license inquiries 987  
 linear algebra 351  
 linear algebraic equation subroutines  
   PDDTSV 518  
   PDDTTRF 532  
   PDDTTRS 548  
   PDGECON 402  
   PDGELS 421  
   PDGEQRF 411  
   PDGESV 355  
   PDGETRF 370  
   PDGETRI 393  
   PDGETRS 381  
   PDGTSV 518  
   PDGTTRF 532  
   PDGTTRS 548  
   PDPBSV 486  
   PDPBTRF 498  
   PDPBTRS 507  
   PDPOCON 476  
   PDPOSV 435  
   PDPOTRF 449  
   PDPOTRI 469  
   PDPOTRS 458  
   PDPTSV 565  
   PDPTTRF 579  
   PDPTTRS 591  
   PZGECON 402  
   PZGELS 421  
   PZGEQRF 411  
   PZGESV 355  
   PZGETRF 370  
   PZGETRI 393  
   PZGETRS 381  
   PZPOCON 476  
   PZPOSV 435  
   PZPOTRF 449  
   PZPOTRI 469  
   PZPOTRS 458  
 sparse (Fortran 77)  
   application program outline 90  
   overview 16, 353  
   PADINIT 643  
   PDGEASB 659  
   PDGEINS 652  
   PDSPASB 655  
   PDSPGIS 664  
   PDSPGPR 661  
   PDSPINIT 645  
   PDSPINS 647  
 sparse (Fortran 90)  
   application program outline 89  
   overview 352  
   PADALL 607  
   PADFREE 635  
   PGEALL 611

linear algebraic equation subroutines  
(*continued*)

sparse (Fortran 90) (*continued*)

PGEASB 622  
PGEFREE 632  
PGEINS 617  
PSPALL 609  
PSPASB 619  
PSPFREE 633  
PSPGIS 627  
PSPGPR 624  
PSPINS 613

linear algebraic equation subroutines,  
sparse 16

linear algebraic equations  
overview 351

linking your program

C programs 92, 96  
C++ programs 93, 96  
dynamic versus static 92, 95  
Fortran programs 92, 95

local arrays, definition of 21

LOCp()

for block ix  
for block-cyclic ix, 28, 32

LOCq()

for block-cyclic ix, 28, 32

logical constant, definition of 991

logical data

conventions xi

logical expression, definition of 991

logical type, definition of 991

long precision

accuracy statement 5  
definition of 991

## M

mailing list for ESSL customers 12

main program, definition of 991

mainframes

definition of 991

mask, definition of 991

math and programming notations xii

math background publications

*See* references, math background

mathematical expressions, conventions  
for xii

mathematical functions, overview 3

matrix 351

array descriptor 27  
block distributing a band matrix 29,  
42  
block distributing a general  
tridiagonal matrix 44  
block distributing a tridiagonal  
matrix 29  
block-cyclically distributing a  
symmetric tridiagonal matrix 30, 48  
definition of 991  
font for xi  
submatrix xi, 28

matrix-matrix product

general matrices, their transposes, or  
their conjugate transposes 239  
real symmetric matrix 256  
real triangular matrix 276, 288

matrix-vector product

general matrix or its transpose 131  
real symmetric matrix 154  
triangular matrix, its transpose, or its  
conjugate transpose 212

max notation and meaning xii

meanings of words in the Glossary 991

messages

list of Parallel ESSL messages 108,  
122

message conventions 107

migrating

array descriptors, for the new release  
of Parallel ESSL 99

your program, to the new  
release/mod of Parallel ESSL 99

min notation and meaning xii

miscellaneous errors

list of messages for 121

mod notation and meaning xii

modification level of Parallel ESSL,  
getting 847

modifying

C programs, for using Parallel ESSL  
for AIX 92

C programs, for using Parallel ESSL  
for Linux 96

C++ programs, for using Parallel ESSL  
for AIX 93

C++ programs, for using Parallel ESSL  
for Linux 96

Fortran programs, for using Parallel  
ESSL for AIX 92

Fortran programs, for using Parallel  
ESSL for Linux 95

modulo notation xii

MPI error messages 107

MPI tasks 77

multiplying

notation xii

## N

name usage restrictions 79

name, definition of 991

names of

eigensystem analysis and singular  
value analysis 675  
Fourier Transforms 797  
Level 2 PBLAS 129  
Level 3 PBLAS 237  
linear algebraic equations 351  
products and acronyms x  
random number generation 837  
utilities 845

National Language Support 102

NLS, National Language Support 102

norm notation xii

notations and conventions xi

notes, conventions used in the subroutine  
descriptions xv

notices 987

number of subroutines in each area 3

numbers

accuracy of computations 5

NUMROC 868

## O

one norm notation xii

one-dimensional data structures,  
distributing 33, 36

online documentation

required Hypertext Markup Language  
products 11

output

accuracy 5

output arguments, conventions used in  
the subroutine descriptions xv

overflow exception, definition of 991

overview

of Parallel ESSL 3

of the documentation 1002

## P

PADALL 607

PADFREE 635

PADINIT 643

Parallel Environment

dynamic linking versus static linking  
when using Parallel ESSL for  
AIX 92

dynamic linking versus static linking  
when using Parallel ESSL for  
Linux 95

how used by Parallel ESSL 4

job processing procedures in 91, 94

Parallel ESSL

*See* ESSL, Parallel (Parallel  
Engineering and Scientific  
Subroutine Library)

Parallel ESSL program number 6

Parallel ESSL release 6

parallel processing

introduction to 4

PARTS, user-supplied subroutine

coding 62

designing 62

using in C programs 63

using in C++ programs 63

PDCFT2 800

PDCFT3 817

PDCRFT2 812

PDCRFT3 831

PDDTSV 518

PDDTTRF 532

PDDTTRS 548

PDF

definition of 991

PDGEASB 659

PDGEBRD 762

PDGECON 402

PDGEHRD 753

PDGEINS 652

PDGELS 421

PDGEMM 239

PDGEMV 131

PDGEQRF 411

PDGER 168

PDGESV 355

PDGESVD 780

PDGETRF 370

PDGETRI 393



- PDGETRS 381
- PDGTSV 518
- PDGTTRF 532
- PDGTRTS 548
- PDLANGE 872
- PDLANSY 879
- PDPBSV 486
- PDPBTRF 498
- PDPBTRS 507
- PDPOCON 476
- PDPOSV 435
- PDPOTRF 449
- PDPOTRI 469
- PDPOTRS 458
- PDPTSV 565
- PDPTTRF 579
- PDPTTRS 591
- PDRCFT2 807
- PDRCFT3 825
- PDSPASB 655
- PDSPGIS 664
- PDSPGPR 661
- PDSPINIT 645
- PDSFINS 647
- PDSYEVX 677
- PDSYGST 739
- PDSYGVX 698
- PDSYMM 256
- PDSYMV 154
- PDSYR 186
- PDSYR2 197
- PDSYR2K 316
- PDSYRK 301
- PDSYTRD 724
- PDTRAN 336
- PDTRMM 276
- PDTRMV 212
- PDTRSM 288
- PDTRSV 224
- PDURNG 839
- performance
  - aspects of parallel processing on the workstations 4
  - coding tips for achieving optimal 77
- PESSL\_ERROR\_SYNC environment variable 77, 103
- PGEALL 611
- PGEASB 622
- PGEFREE 632
- PGEINS 617
- pi notation xii
- precision, short and long 5
- preconditioning a sparse matrix 624, 661
- primary, definition of 991
- problems, handling
  - See diagnosis procedures
- problems, IBM support for 101
- procedures, job processing
  - setting up your own AIX 91
  - setting up your own Linux 94
- process grid, definition of ix, 21
- processing your program
  - requirements for Parallel ESSL 6
  - steps involved in 91, 94
  - using parallel subroutines on the workstations 4

- product
  - matrix-matrix
    - general matrices, their transposes, or their conjugate transposes 239
    - real symmetric matrix 256
    - real triangular matrix 276, 288
  - matrix-vector
    - general matrix or its transpose 131
    - real symmetric matrix 154
    - triangular matrix, its transpose, or its conjugate transpose 212
- product names, acronyms for x
- product names, using x
- products, programming
  - required by Parallel ESSL for Linux, programming 8
  - required by Parallel ESSL, programming 7
- program
  - communication errors 106
  - computational errors 105
  - distributing your data across processes 21
  - errors on the workstations 104
  - ESSL error messages 107
  - input-argument errors 104
  - miscellaneous errors 106
  - MPI error messages 107
  - resource errors 105
  - using error handling 101
- program exceptions
  - definition of 991
  - description of Parallel ESSL on the workstations related 104
- program number for Parallel ESSL 1002
- program number of Parallel ESSL 6
- programming items, font for xi
- programming products
  - required by Parallel ESSL 7
  - required by Parallel ESSL for Linux 8
- PSCFT2 800
- PSCFT3 817
- PSCRFT2 812
- PSCRFT3 831
- PSPALL 609
- PSPASB 619
- PSPFREE 633
- PSPGIS 627
- PSPGPR 624
- PSPINS 613
- PSRCFT2 807
- PSRCFT3 825
- PTF
  - definition of 991
  - getting the most recent level applied 847
- publications
  - list of Parallel ESSL 1002
  - math background 999
- PZGEBRD 762
- PZGECON 402
- PZGELS 421
- PZGEMM 239
- PZGEMV 131

- PZGEQRF 411
- PZGERC 168
- PZGERU 168
- PZGESV 355
- PZGESVD 780
- PZGETRF 370
- PZGETRI 393
- PZGETRS 381
- PZHEEVX 677
- PZHEGST 739
- PZHEGVX 698
- PZHEMM 256
- PZHEMV 154
- PZHER 186
- PZHER2 197
- PZHER2K 316
- PZHETRD 724
- PZLANGE 872
- PZLANHE 879
- PZLANSY 879
- PZPOCON 476
- PZPOSV 435
- PZPOTRF 449
- PZPOTRI 469
- PZPOTRS 458
- PZSYMM 256
- PZSYR2K 316
- PZTRANC 336
- PZTRANU 336
- PZTRMM 276
- PZTRMV 212
- PZTRSM 288
- PZTRSV 224

## R

- random number generation
  - data distribution 36
  - overview 837
  - PDURNG 839
  - uniformly distributed 839
- rank-2k update
  - real symmetric matrix 197, 316
- rank-k update
  - real symmetric matrix 186, 301
- rank-one update
  - general matrix 168
- real constant, definition of 991
- real data
  - conventions xi
- real general matrix eigensystem analysis and singular value analysis
  - subroutine 675
- real symmetric matrix eigensystem analysis and singular value analysis
  - subroutine 675
- real type, definition of 991
- reference information
  - math background texts and reports 999
  - organization of 127
  - what is in each subroutine description and the conventions used xiii
- references, math background
  - texts, journals, reports 999
- release of Parallel ESSL 6
- release of Parallel ESSL, getting 847

- reporting problems to IBM 101
- required publications 1002
- required software products 7
- requirements
  - for Parallel ESSL 6
  - hardware 6
  - software products 7, 8
  - system software 6
- resource-allocation errors
  - list of messages for 120
  - overview 105
  - reducing memory constraints 105
- restrictions, Parallel ESSL coding 79
- results
  - accuracy 5
- routine names 79
- Run-Time Environment, XL Fortran 8
- running your program
  - C programs 92, 96
  - C++ programs 93, 96
  - Fortran programs 92, 95

## S

- sample programs
  - using Fortran 77 sparse subroutines 669, 941
  - using Fortran 90 sparse subroutines 637, 932, 950
- sample programs, thermal diffusion 895
- sample utilities 895
- scalar data
  - conventions xi
- scalar items, font for xi
- scalar, definition of 991
- scope, definition of ix
- sequences
  - three-dimensional, distributing 69
  - two-dimensional, distributing 64
- service, IBM 101
- setting up, AIX procedures 91
- setting up, Linux procedures 94
- shape of an array, definition of 991
- short precision
  - accuracy statement 5
  - definition of 991
- SIGN notation and meaning xii
- sin notation xii
- single block matrix, distributing 40
- size notation xii
- size of array
  - definition of 991
- software for linking, loading, or running 9
- software products
  - required by Hypertext Markup Language 11
- software products required 7, 8
- solving
  - complex Hermitian matrix 476
  - general matrix 381, 393, 402, 469
  - general tridiagonal matrix 518, 548
  - positive definite complex Hermitian matrix 458
  - positive definite real symmetric matrix 458

- solving (*continued*)
  - positive definite real symmetric or complex Hermitian matrix 476
  - positive definite symmetric band matrix 486, 507
  - positive definite symmetric tridiagonal matrix 565, 591
  - real symmetric matrix 476
  - sparse matrix 627, 664
  - triangular matrix 224, 288
- sparse linear algebraic equation subroutines (Fortran 90) 16
- sparse matrix, distributing 62
- spectral norm notation xii
- square root notation xii
- statement function, definition of 991
- statement label, definition of 991
- statement number, definition of 991
- statement, definition of 991
- storage mode, FFT-packed 65, 70
- stride
  - definition of 991
- structures of data, distributing across processes 21
- submatrix
  - notation xi
  - specifying 28
- subprogram
  - definition of ix, 991
  - linear algebra 129, 237
  - meaning of ix
- subroutine
  - conventions used in the description of xiii
  - definition ix
  - overview of Parallel ESSL 3
- subroutines, Parallel ESSL
  - DESCINIT 849
  - DESCSET 852
  - ICEIL 855
  - ILCM 856
  - INDXG2L 857
  - INDXG2P 859
  - INDXL2G 861
  - INFOG1L 863
  - INFOG2L 865
  - IPESSL 847
  - NUMROC 868
  - PADALL 607
  - PADFREE 635
  - PADINIT 643
  - PDCFT2 800
  - PDCFT3 817
  - PDCRFT2 812
  - PDCRFT3 831
  - PDDTSV 518
  - PDDTTRF 532
  - PDDTTRS 548
  - PDGEASB 659
  - PDGEBRD 762
  - PDGECON 402
  - PDGEHRD 753
  - PDGEINS 652
  - PDGELS 421
  - PDGEMM 239
  - PDGEMV 131
  - PDGEQRF 411

- subroutines, Parallel ESSL (*continued*)
  - PDGER 168
  - PDGESV 355
  - PDGESVD 780
  - PDGETRF 370
  - PDGETRI 393
  - PDGETRS 381
  - PDGTSV 518
  - PDGTTRF 532
  - PDGTTRS 548
  - PDLANGE 872
  - PDLANSY 879
  - PDPBSV 486
  - PDPBTRF 498
  - PDPBTRS 507
  - PDPOCON 476
  - PDPOSV 435
  - PDPOTRF 449
  - PDPOTRI 469
  - PDPOTRS 458
  - PDPTSV 565
  - PDPTTRF 579
  - PDPTTRS 591
  - PDRCFT2 807
  - PDRCFT3 825
  - PDSPASB 655
  - PDSPGIS 664
  - PDSPGPR 661
  - PDSPINIT 645
  - PDSPINS 647
  - PDSYEVX 677
  - PDSYGST 739
  - PDSYGVX 698
  - PDSYMM 256
  - PDSYMV 154
  - PDSYR 186
  - PDSYR2 197
  - PDSYR2K 316
  - PDSYRK 301
  - PDSYTRD 724
  - PDTRAN 336
  - PDTRMM 276
  - PDTRMV 212
  - PDTRSM 288
  - PDTRSV 224
  - PDURNG 839
  - PGEALL 611
  - PGEASB 622
  - PGEFREE 632
  - PGEINS 617
  - PSCFT2 800
  - PSCFT3 817
  - PSCRFT2 812
  - PSCRFT3 831
  - PSPALL 609
  - PSPASB 619
  - PSPFREE 633
  - PSPGIS 627
  - PSPGPR 624
  - PSPINS 613
  - PSRCFT2 807
  - PSRCFT3 825
  - PZGEBRD 762
  - PZGECON 402
  - PZGELS 421
  - PZGEMM 239
  - PZGEMV 131

subroutines, Parallel ESSL (*continued*)

PZGEQRF 411  
PZGERC 168  
PZGERU 168  
PZGESV 355  
PZGESVD 780  
PZGETRF 370  
PZGETRI 393  
PZGETRS 381  
PZHHEVX 677  
PZHEGST 739  
PZHEGVX 698  
PZHEMM 256  
PZHEMV 154  
PZHER 186  
PZHER2 197  
PZHER2K 316  
PZHERK 301  
PZHETRD 724  
PZLANGE 872  
PZLANSY 879  
PZPOCON 476  
PZPOSV 435  
PZPOTRF 449  
PZPOTRI 469  
PZPOTRS 458  
PZSYMM 256  
PZSYR2K 316  
PZSYRK 301  
PZTRANC 336  
PZTRANU 336  
PZTRMM 276  
PZTRMV 212  
PZTRSM 288  
PZTRSV 224

subscript expression, definition of 991  
subscript notation, what it means xii  
subscript, definition of 991  
summation notation xii  
superscript notation, what it means xii  
support, IBM 101  
symbols, special usage of xii  
syntax, conventions used in the  
    subroutine descriptions xiii  
system software required 6

## T

termination, program  
    communication errors 106  
    computational errors 105  
    ESSL error messages 107  
    input-argument errors 104  
    miscellaneous errors 106  
    MPI error messages 107  
    on the workstations 104  
    resource errors 105  
terminology in the Glossary 991  
terminology, names of products x  
textbooks cited  
    *See* references, math background  
times notation, multiply xii  
trademarks 988  
transpose  
    notation xii  
    of a matrix inverse notation xii  
    of a vector or matrix notation xii

tridiagonal matrix, distributing 48  
two-dimensional data structures,  
    distributing 40, 53  
type declaration, definition of 991  
type font usage xi

## U

underflow  
    definition of 991  
uniformly distributed random numbers,  
    generate 839  
user-supplied subroutine 62  
using this book vii, viii  
utility subroutines 845  
    DESCINIT 849  
    DESCSET 852  
    ICEIL 855  
    ILCM 856  
    INDXG2L 857  
    INDXG2P 859  
    INDXL2G 861  
    INFOG1L 863  
    INFOG2L 865  
    IPESSL 847  
    NUMROC 868  
    PDLANGE 872  
    PDLANSY 879  
    PZLANGE 872  
    PZLANHE 879  
    PZLANSY 879

## V

variable, definition of 991  
vector  
    array descriptor 27  
    definition of 991  
    distributing data for 33  
    font for xi  
    special form of submatrix xi  
    submatrix 28  
version of Parallel ESSL, getting 847  
versions of subroutines 3

## W

warnings  
    list of messages for 120  
words in the Glossary 991  
workstations  
    definition of 991  
    required for Parallel ESSL 6

## X

XL C/C++ 7  
XL Fortran 7  
XL Fortran Run-Time Environment 8



---

## Readers' Comments — We'd Like to Hear from You

Parallel Engineering and Scientific Subroutine Library  
for AIX, Version 3 Release 2, and  
Parallel Engineering and Scientific Subroutine Library  
for Linux on POWER, Version 3 Release 2  
Guide and Reference

Publication No. SA22-7906-03

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Complete	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to find	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to understand	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Well organized	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Applicable to your tasks	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

---

Name

---

Address

---

Company or Organization

---

Phone No.



Cut or Fold  
Along Line

Fold and Tape

Please do not staple

Fold and Tape



NO POSTAGE  
NECESSARY  
IF MAILED IN THE  
UNITED STATES

**BUSINESS REPLY MAIL**

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation  
Department 55JA, Mail Station P384  
2455 South Road  
Poughkeepsie, NY 12601-5400



Fold and Tape

Please do not staple

Fold and Tape

Cut or Fold  
Along Line





Program Number: 5765-F84 and 5765-G18

SA22-7906-03

