LAPACK Working Note 93
Installation Guide for ScaLAPACK[1]

J. Choi, J. J. Dongarra, L. S. Ostrouchov, A. Petitet, and R. C. Whaley
Department of Computer Science
University of Tennessee
Knoxville, Tennessee 37996-1301

and

J. Demmel, I. Dhillon, and K. Stanley
Computer Science Division
University of California, Berkeley
Berkeley, CA 94720

and

D. Walker
Mathematical Sciences Section
Oak Ridge National Laboratory
P.O. Box 2008, Bldg. 6012
Oak Ridge, TN 37831-6367

VERSION 1.0, February 28, 1995
DATE: March, 1995

Abstract

This working note describes how to install and test version 1.0 of ScaLAPACK. These two-dimensional distributed memory versions of common LAPACK routines rely on calls to the BLAS for local computation, and calls to the PBLAS for global computations. For portability concerns, communication takes place inside the PBLAS through calls to the BLACS. The design of the testing/timing programs for the ScaLAPACK codes is also discussed.

Only a subset of the routines are available. Routines that are NOT available yet are the least squares driver, condition estimation and iterative refinement for LU and Cholesky (so the expert drivers are not ready), PSIGN, and PUMMA. Please disregard discussion of these routines in this installation guide. They will be available soon.

[1]This work was supported in part by the National Science Foundation Grant No. ASC-9005933; by the Defense Advanced Research Projects Agency under contract DAAL03-91-C-0047, administered by the Army Research Office; by the Office of Scientific Computing, U.S. Department of Energy, under Contract DE-AC05-84OR21400; and by the National Science Foundation Science and Technology Center Cooperative Agreement No. CCR-8809615.

Contents

1 Introduction

Only a subset of the routines are available. Routines that are NOT available yet are the least squares driver, condition estimation and iterative refinement for LU and Cholesky (so these computational routines and the associated expert drivers are not ready), PSIGN, and PUMMA. Please disregard discussion of these routines in this installation guide. They will be available soon.

This working note describes how to install and test version 1.0 of ScaLAPACK[7]. The supported platforms are: PVM 3.3, the Intel i860 series, Thinking Machines CM-5, or the IBM SP series. (It is very important to note that only PVM version 3.3 is supported with the BLACS[19, 3]. Due to major changes in PVM and the resulting changes required in the BLACS, earlier versions of PVM are NOT supported. If you have a previous release of PVM you must obtain version 3.3 to install and test the BLACS. See section 3.2 for details of how to obtain the latest release of PVM.)

Section 2 describes the distribution and organization of the files. Step-by-step installation and testing/timing instructions appear in Section 3. For users desiring additional information, Section 4 gives details on the testing/timing programs for the ScaLAPACK codes and their input files. Appendices A and B describe the ScaLAPACK driver, computational, and auxiliary routines currently available.

2 Overview of Distribution Format and Contents

The software is distributed in the form of a compressed tar file which contains the PBLAS source and test code, PUMMA source and test code, and the ScaLAPACK source and test codes. A TOOLS directory is also supplied which contains needed LAPACK[6, 2, 1] routines as well as support routines.

It is assumed that you have the BLACS, BLAS[18, 15, 14], and PVM (if necessary) available on your machine. If this is not the case, you MUST obtain the missing component from netlib and have the library available for the ScaLAPACK installation. The BLACS are available in the blacs directory on netlib; the Fortran BLAS are available in the blas directory on netlib; and, PVM 3.3 is available in the pvm3 directory on netlib. Refer to the index in the appropriate directory for further details.

To obtain the complete ScaLAPACK package, send email to `netlib@ornl.gov` and in the message type:

```
send index from scalapack
```

It is also possible to obtain selected pieces of the SCALAPACK directory structure without having to download the entire package. At this time, one may obtain individual tar files of the PBLAS or PUMMA. Refer to the scalapack index on netlib for further details.

The software in the `tar` file is organized in a number of directories as shown in Figure 1. Please note that this figure does not reflect everything that is contained in the `SCALAPACK` directory. Input and instructional files are also located at various levels. Each of the lowest

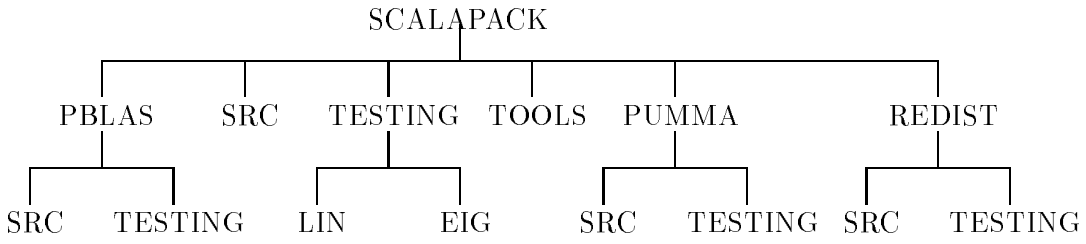

Figure 1: Organization of ScaLAPACK

level directories in the tree structure contains a makefile to create a library or a set of executable programs for testing/timing. Libraries are created in the `SCALAPACK` directory and executable files are created in the `TESTING` directory(ies). Input files are copied into the `TESTING` directory at the time each executable is created.

The PBLAS are parallel versions of the Level 1, 2, and 3 BLAS. For more details on the PBLAS, refer to [12, 11]. The same naming scheme as the BLAS has been maintained in the PBLAS with the addition of the prefix `P` to signify that it is a PBLAS routine. All precisions – REAL, DOUBLE PRECISION, COMPLEX, and COMPLEX*16 – of the routines are available.

All precisions of ScaLAPACK routines are also available with this release, with the exception of the eigenproblem routine PDSYEVX. Future releases will include single precision real and double precision real of the symmetric eigenproblem as well as the singular value decomposition.

2.1 ScaLAPACK Routines

Like LAPACK, there are three classes of ScaLAPACK routines:

- **driver** routines solve a complete problem, such as solving a system of linear equations or computing the eigenvalues of a real symmetric matrix. Please refer to Appendix A for a list of all available driver routines. Global and local input error-checking are performed for these routines.

- **computational** routines, also called simply ScaLAPACK routines, perform a distinct computational task, such as computing the LU decomposition of an m-by-n matrix, or reducing a real general matrix to upper Hessenberg form. Please refer to Appendix A for a list of all available ScaLAPACK computational routines. Global and local input error-checking are performed for these routines.

- **auxiliary** routines are all of the other subroutines called by the computational routines. Among them are subroutines to perform subtasks of block algorithms, and a number of routines to perform common low-level computations. A list of all available ScaLAPACK auxiliary routines can be found in Appendix B. In general, no input error-checking is performed on the auxiliary routines. The exception to this rule is for the auxiliary routines which are Level 2 equivalents of computational routines (e.g., PxGETF2, PxGEQR2, PxORMR2, PxORM2R, etc.). For these routines, local input error-checking routines is performed.

LAPACK auxiliary routines are also used whenever possible for local computation.

2.2 Testing/Timing Routines

Testing/timing programs are included for each of the ScaLAPACK routines. Refer to section 4 for more details.

3 Installation Procedure

Installing, testing, and timing ScaLAPACK involves the following steps:

1. Build the PVM 3.3 library, if necessary.
2. Build the BLACS library.
3. Build the BLAS library, if necessary.
4. Uncompress and untar the file.
5. Edit the SLmake.inc include file.
6. Edit the top-level Makefile, and type `make`.
7. Run the PB-BLAS Test Suite.
8. Run the REDIST Test Suite.
9. Run the ScaLAPACK Test Suite.
10. Communicate any difficulties to the authors.

3.1 Build the PVM 3.3 Library

If you wish to use ScaLAPACK on PVM, you will need to have PVM 3.3 installed on your machine. If you do not already have PVM installed on your machine, you need to obtain the source code from netlib by sending email to `netlib@ornl.gov` and in the message type: `send index from pvm3`. This index contains instructions on how to obtain the source code. After obtaining the source, follow the instructions in the PVM User's Guide [17] for installation.

NOTE: Any version of PVM previous to 3.3 is not supported with the BLACS.

3.2 Build the BLACS Library

If you wish to use ScaLAPACK, you MUST have an appropriate version of the BLACS installed on your machine. If you do not already have the BLACS installed on your machine, you need to obtain the source code from netlib or via the html page (see below). Refer to the blacs index on netlib or the BLACS html page for further details. The html page also contains a troubleshooting section and detailed information on each individual BLACS routine. After obtaining the source, follow the instructions in "A User's Guide to the BLACS" or in the "Installing the BLACS" section of the html page to install the library. Instructions for running the BLACS Test Suite can be found in "A User's Guide to the BLACS Tester". Both of these documents are available via the URL

http://www.cs.utk.edu/~rwhaley/Blacs.html

3.3 Build the BLAS Library

Ideally, a highly optimized version of the BLAS library already exists on your machine. You may already have a library containing some of the BLAS, but not all (Level 1 and 2, but not Level 3, for example). If so, you should use your local version of the BLAS wherever possible.

If you do not already have the BLAS installed on your machine, you need to obtain the source code from netlib by sending email to `netlib@ornl.gov` and in the message type: `send index from blas`. This index contains instructions on how to obtain the source code. After obtaining the source, follow the instructions in the LAPACK Installation Guide [4] for installation.

3.4 Uncompress and untar the file

To unpack the `scalapack.tar.Z`, type the following command:

```
uncompress -c scalapack.tar.Z | tar xvf -
```

This will create a top-level directory called `SCALAPACK`, with the rest of the files in the directory structure as previously discussed. You will need approximately 7 Mbytes of space for the tar file.

Your total space requirements will vary depending upon if all platforms of the BLACS are installed and the size of executable files that your configuration can handle.

3.5 Edit the SLmake.inc include file

Example machine-specific `SCALAPACK/SLmake.inc` files are provided in the top-level `SCALAPACK` directory for the PVM, Intel, CM-5, and SP series implementations. When you have selected the machine to which you wish to install ScaLAPACK, copy the appropriate sample include file (if one is present) into `SCALAPACK/SLmake.inc`. For example, if you wish to run ScaLAPACK on a DEC ALPHA,

```
cp SLmake.alpha SLmake.inc
```

HOWEVER, SLIGHT MODIFICATIONS TO THE INCLUDE FILE WILL STILL NEED TO BE MADE.

Go to `SCALAPACK` and edit the `SLmake.inc` make include file to contain the following:

1. Specify the complete path to the top level `SCALAPACK` directory called `home`.
2. Identify the platform to which you will be installing the libraries. If your directory structure for ScaLAPACK is different than the aforementioned structure, you will also need to specify locations of `SCALAPACK` subdirectories.

3. Define F77, NOOPT, F77FLAGS, CC, CCFLAGS, LOADER, LOADFLAGS, ARCH, ARCHFLAGS, and RANLIB, to refer to the compiler and compiler options, loader and loader options, library archiver and options, and ranlib for your machine. If your machine does not have ranlib set RANLIB = echo.

4. Specify the C preprocessor definitions for compilation, BLACSDBGLVL and CDEFS. The possible values for BLACSDBGLVL are 0 and 1. The possible options for CDEFS are -DAdd_, -DNoChange, and -DUPCASE. If you are on a DEC ALPHA, you must also add -DNO_IEEE to the definition of CDEFS.

5. Specify the locations of the needed libraries: BLACS, BLAS, and PVM.

This make include file is referenced inside each of the makefiles in the various subdirectories. As a result, there is no need to edit the makefiles in the subdirectories. All information that is machine specific has been defined in this include file. Do not be alarmed when you see multiple copies of SLmake.inc sprinkled throughout various subdirectories. These multiple copies are included because portions of the distribution are available individually. By default, all makefiles point to the SLmake.inc in the top-level SCALAPACK directory.

3.6 Top-Level ScaLAPACK Makefile

A top-level SCALAPACK makefile has been included to build all libraries and testing executables. This makefile is very useful if you are familiar with the installation process and wish to do a quick installation. Your instructions to build all libraries and testing executables are:

```
cd SCALAPACK

make
```

Alternatively, if you wish to only build the libraries, you can specify

```
make lib.
```

Or, if you wish to only build the executables (assuming that all libraries have previously been built)

```
make exe.
```

If you wish to build only selected libraries or executables, you can modify the lib or exe definition accordingly.

To specify the data types to be built, you will need to modify the definition of PRECISIONS. By default, PRECISIONS is set to

```
PRECISIONS = single double complex complex16
```

to build all precisions of the libraries and executables. If you only wish to compile the single precision real version of a target specify `single`, for double precision real specify `double`, for single precision complex specify `complex`, and for double precision complex specify `complex16`.

By default, the presence of no arguments following the `make` command will result in the building of all data types. The make command can be run more than once to add another data type to the library if necessary.

You may then proceed to running each of the individual test suites. See section 3.7 for details on the PB-BLAS Test Suite, section 3.8 to run the REDIST test suite, and section 3.9 for details on the ScaLAPACK Test Suite. After all testing has been completed, you can remove all object files from the various subdirectories and all executables from the `SCALAPACK/TESTING` directory by typing

```
make clean.
```

Or, you can selectively remove only the object files with `make cleanlib`, or `make cleanexe` to remove only the executable files.

Build the TOOLS Library

a) Go to the directory `SCALAPACK/TOOLS`.

b) Type `make`.

By default, the TOOLS library is created in the top level directory `SCALAPACK`.

Build the PBLAS Library

a) Go to the directory `SCALAPACK/PBLAS/SRC`.

b) Type `make`.

By default, the PBLAS library is created in the top level `SCALAPACK` directory.

Build the REDIST Library

a) Go to the directory `SCALAPACK/REDIST/SRC`.

b) Type `make`.

By default, the REDIST library is created in the top level `SCALAPACK` directory.

3.7 Run the PB-BLAS Test Suite

At the present time, a tester for the PBLAS is not available. We are instead including the tester for the PB-BLAS which are internal to the PBLAS. The PBLAS tester will be included with the next release.

a) Go to the directory `SCALAPACK/PBLAS/TESTING`.

b) Type `make` followed by the data types desired. For the Level 2 PB-BLAS routines, the testing executables are called `xspb2chk`, `xdpb2chk`, `xcpb2chk`, and `xzpb2chk`, and are created in the `PBLASTSTdir` directory as defined in `SLmake.inc`. Likewise, the testing executables for the Level 3 PB-BLAS are `xspb3chk`, `xdpb3chk`, `xcpb3chk`, and `xzpb3chk`. There is one input file associated with each testing executable. For example, the input file for `xspb2chk` is called `PBS2BLAS.dat`. The input files are copied to the `PBLASTSTdir` directory at the time the executables are built.

c) Do one of the following for the platform to which you have installed the BLACS:

Instructions using the PVM BLACS

First, insure that the PVM library and tester executable files have been compiled for each of the machines used in your PVM implementation. PVM 3.3 requires that executable files be stored in a particular directory so that the PVM daemon can find them. In the general case, PVM looks for executable files in `~/pvm3/bin/`*arch*, where *arch* specifies the architecture for which the executable has been built. For example, if one wished to run the test program on a SUN SPARCstation and on an IBM RS6000 workstation, appropriately compiled executable files need to be placed in `~/pvm3/bin/SUN4` and `~/pvm3/bin/RS6K` (for more directory information, consult the PVM documentation). If you wish to run the tests on machines that are not connected to the same file system, you need to make sure that the executable is available on each file system. Next, start pvm by typing

```
pvm
```

At this point, you specify the machines that are to take part in the testing process (see the PVM documentation for more information). Finally, to test the REAL PVM Level 2 PB-BLAS, start the test program by typing:

```
xspb2chk
```

on one of the machines that is a member of your PVM machine. This program will then instruct the PVM daemon to start processes on the other computers in your PVM machine and you will be prompted by the program for the name of the executable. Make sure that `PBS2BLAS.dat` is located in the same directory as `xspb2chk`. It is read on the machine from which you type `xspb2chk` and its contents distributed to the other computers in your PVM machine.

Alternatively, you can use `blacs_setup.dat` *to perform much of this process. This file specifies the name of the executable and the machines to spawn in your pvm cluster, as well as a few other features. See the "A User's Guide to the BLACS" for details. However, the use of this file is not recommended for the naive user.*

Similar commands should be used for the other test programs, with the second letter 's' in the executable and data file replaced by 'd', 'c', or 'z'. The name of the output file is indicated on the first line of the input file and is currently defined to be `PBBLAS.out`

for the REAL version, with similar names for the other data types. The user may also choose to send all output to standard error.

Instructions using the Intel BLACS

If your compiler and loader are located on a separate machine and your current disk and directory is not cross-mounted or NFS-mounted on the destination computer, you will need to first transfer the needed testing executable files and the input files to your Intel computer. You must then decide on the number of processors to be allocated for your testing/timing runs. In each input file, there is a set of processor grid dimensions specified (`VALUES OF P` and `VALUES OF Q`). The number of processors you allocate for an Intel class machine must be greater than or equal to the maximum processor grid (`P` $\times$ `Q`) specified in the input file. The default input files assume 8 processors. Then, to run the tests of the REAL Level 2 PB-BLAS routines, for example, change directories to `PBLASTSTdir`, edit the appropriate `gamma.exe`, `delta.exe`, or `pgon.exe` file to load the executable `xspb2chk`. And type,

`gamma.exe` *[number of processors to allocate]*

or

`delta.exe` *[number of processor rows] [number of processor columns]*

or

`pgon.exe` *[number of processors to allocate]*

depending upon which Intel computer you are using, and omitting the square brackets.

Similar commands should be used for the other test programs, with the second letter 's' in the executable replaced by 'd', 'c', or 'z'. The name of the output file is indicated on the first line of the input file and is currently defined to be `PBBLAS.out` for the REAL version, with similar names for the other data types. The user may also choose to send all output to standard error.

Instructions using the CM-5 BLACS

To test the REAL CM-5 Level 2 PB-BLAS, for example, start the test program by typing:

```
xspb2chk
```

Similar commands should be used for the other test programs, with the second letter 's' in the executable and data file replaced by 'd', 'c', or 'z'. The name of the output file is indicated on the first line of the input file and is currently defined to be `PBBLAS.out` for the REAL version, with similar names for the other data types. The user may also choose to send all output to standard error.

If the tests were not successful, and you cannot easily determine the cause, please contact the authors as directed in Section 3.10. Please tell us the types of machines on which the tests were run, the compiler and compiler options that were used, details of the BLACS library that you used, and a copy of the input file (for example, `PBS2BLAS.dat`) that was used.

Instructions using the SP BLACS

If your compiler and loader are located on a separate machine and your current disk and directory is not cross-mounted or NFS-mounted on the destination computer, you will need to first transfer the needed testing executable files and input files from the `SCALAPACK/TESTING` directory to your IBM SP-1 or SP-2 computer. You must then decide on the number of processors to be allocated for your testing/timing runs. In each input file, there is a set of processor grid dimensions specified (`VALUES OF P` and `VALUES OF Q`). The number of processors you allocate for an SP series machine must be greater than or equal to the maximum processor grid (`P` $\times$ `Q`) specified in the input file. The default input files assume 8 processors.

Then, for example, to test the REAL Level 2 PB-BLAS, edit the `sp1.exe` file to load the executable `xspb2chk`. And type,

`sp1.exe` *[number of processors to allocate]*

omitting the square brackets.

Please note that the commands needed to load and run the executable are not standard across all SP series machines, the included script file `sp1.exe` runs the tests on an IBM SP-1 located at Cornell University.

Similar commands can be used for alternate precisions of the same test program or other test programs, where, `xdlu` is replaced by `xdls`, `xdqr`, `xdllt`, `xdhrd`, `xdtrd`, `xdbrd`, `xdtri`, `xdpsign`, or `xdsep`. The name of the output file is indicated on the third line of the input file and is currently defined to be `lu.out` for the LU tester, with similar names for the other data types. The user may also choose to send all output to standard error.

3.8 Run the REDIST Test Suite

The redistribution routines are still under development. They allow the redistribution of 2-D block cyclic distributed general or trapezoidal matrix from an arbitrary $P \times Q$ grid with arbitrary blocksize to another grid with arbitrary blocksize. At the present time, these routines are not used in the ScaLAPACK library but they will be integrated into the library in the future.

a) Go to the directory `SCALAPACK/REDIST/TESTING`.

b) Type `make` followed by the data types desired. The testing executables are called `xigemr`, `xsgemr`, `xdgemr`, `xcgemr`, `xzgemr` for the redistribution of general matrices.

They are called `xitrmr`, `xstrmr`, `xdtrmr`, `xctrmr`, `xztrmr`. `xdpb2chk`, `xcpb2chk`, and `xzpb2chk` for trapezoidal matrices, and are created in the `REDISTdir/TESTING` directory as defined in `SLmake.inc`. There is one input file `GEMR2D.dat` for general matrices, and one input file `TRMR2D.dat` for trapezoidal matrices. Each line of the input file is a separate test.

Build the ScaLAPACK Library

To build the entire ScaLAPACK Library, do the following:

a) Go to the directory `SCALAPACK/SRC`.

b) Type `make` followed by the data types desired.

The ScaLAPACK library is created in the top level `SCALAPACK` directory.

Build the ScaLAPACK Testing Executables

If you wish to build all ScaLAPACK testing executables, do the following:

a) Go to the directory `SCALAPACK/TESTING/LIN`.

b) Type `make` followed by the data types desired.

c) Go to the directory `SCALAPACK/TESTING/EIG`.

d) Type `make` followed by the data types desired.

The executables are deposited into the `TESTINGdir` directory as specified in `SLmake.inc`. The individual input files are copied into the `TESTINGdir` directory after each executable is built.

Alternatively, if you do not wish to run all of the separate tests, you can modify the definition of `all` in the `SCALAPACK/TESTING/LIN` or `SCALAPACK/TESTING/EIG` directory to specify only the desired executable.

3.9 Run the ScaLAPACK Test Suite

There are ten distinct test programs for testing the ScaLAPACK routines of the following type: LU, Cholesky, QR (RQ, LQ, QL, and QP), Linear Least Squares, upper Hessenberg reduction, tridiagonal reduction, bidiagonal reduction, triangular inversion, PSIGN (parallel SIGN), and the symmetric eigenproblem. Each of the test programs is automatically timed and reports a table of execution times and megaflop rates. There is one input file for each test program. As previously stated, the input files reside in the `SCALAPACK/TESTING` subdirectory and are copied into the `TESTINGdir` directory (as specified in the `SLmake.inc` file) at the time the executables are built. All testing programs occur in four precisions, with the exception of the symmetric eigenproblem which only occurs in SINGLE and DOUBLE PRECISION REAL. For more information on the test programs and how to modify the input files see Section 4.

a) Do one of the following:

Instructions using the PVM BLACS

First, insure that the PVM library and tester executable files have been compiled for each of the machines used in your PVM implementation. PVM 3.3 requires that executable files be stored in a particular directory so that the PVM daemon can find them. In the general case, PVM looks for executable files in `~/pvm3/bin/`*arch*, where *arch* specifies the architecture for which the executable has been built. If you wish to run the tests on machines that are not connected to the same file system, you need to make sure that the executable is available on each file system. Next, start pvm by typing

```
pvm
```

At this point, you specify the machines that are to take part in the testing process (see the PVM documentation for more information).

Finally, to test the DOUBLE PRECISION ScaLAPACK LU routines, start the test program by typing:

```
xdlu
```

on one of the machines that is a member of your PVM machine. This program will then instruct the PVM daemon to start processes on the other computers in your PVM machine and you will be prompted by the program for the name of the executable. Make sure that `LU.dat` is located in the same directory as `xdlu`. It is read on the machine from which you type `xdlu` and its contents distributed to the other computers in your PVM machine.

Similar commands can be used for alternate precisions of the same test program or other test programs, where, for example in double precision, `xdlu` is replaced by `xdls`, `xdqr`, `xdllt`, `xdhrd`, `xdtrd`, `xdbrd`, `xdtri`, `xdpsign`, or `xdsep`. The name of the output file is indicated on the first line of the input file and is currently defined to be `lu.out` for the LU tester, with similar names for the other data types. The user may also choose to send all output to standard error.

Instructions using the Intel BLACS

If your compiler and loader are located on a separate machine and your current disk and directory is not cross-mounted or NFS-mounted on the destination computer, you will need to first transfer the needed testing executable files `xdlu`, `xdqr`, `xdllt`, `xdhrd`, `xdtrd`, `xdbrd`, and `xdtri`, and the input files `LU.dat`, `QR.dat`, `LLT.dat`, `HRD.dat`, `TRD.dat`, `BRD.dat`, `TRI.dat`, and/or `PSIGN.dat`, from the `SCALAPACK/TESTING` directory to your Intel computer. As with the PB-BLAS tester, you must then decide on the number of processors to be allocated for your testing/timing runs. In each input file, there is a set of processor grid dimensions specified (`VALUES OF P` and `VALUES OF Q`). The number of processors you allocate for an Intel class machine must be greater

than or equal to the maximum processor grid (`P` $\times$ `Q`) specified in the input file. The default input files assume 8 processors.

Then, for example, to run the tests of the DOUBLE PRECISION REAL LU routines, edit the appropriate `gamma.exe`, `delta.exe`, `pgon.exe` file to load the executable `xdlu`. And type,

`gamma.exe` *[number of processors to allocate]*

or

`delta.exe` *[number of processor rows] [number of processor columns]*

or

`pgon.exe` *[number of processors to allocate]*

depending upon which Intel computer you are using, and omitting the square brackets.

Similar commands can be used for alternate precisions of the same test program or other test programs, where, `xdlu` is replaced by `xdls`, `xdqr`, `xdllt`, `xdhrd`, `xdtrd`, `xdbrd`, `xdtri`, `xdpsign`, or `xdsep`. The name of the output file is indicated on the third line of the input file and is currently defined to be `lu.out` for the LU tester, with similar names for the other data types. The user may also choose to send all output to standard error.

Instructions using the CM-5 BLACS

To test the DOUBLE PRECISION ScaLAPACK LU routines, for example, start the test program by typing:

`xdlu`

Similar commands can be used for alternate precisions of the same test program or other test programs, where, `xdlu` is replaced by `xdls`, `xdqr`, `xdllt`, `xdhrd`, `xdtrd`, `xdbrd`, `xdtri`, `xdpsign`, or `xdsep`. If errors were detected, please contact the authors as directed in Section 3.10. Please tell us the type of machine on which the tests were run, the compiler and compiler options that were used, details of the BLACS library that you used, and a copy of the input file (for example, `LU.dat`) that was used. For more details on the test program, see section 4.

Instructions using the SP BLACS

If your compiler and loader are located on a separate machine and your current disk and directory is not cross-mounted or NFS-mounted on the destination computer, you will need to first transfer the needed testing executable files `xdlu`, `xdqr`, `xdllt`, `xdhrd`, `xdtrd`, `xdbrd`, and `xdtri`, and the input files `LU.dat`, `QR.dat`, `LLT.dat`, `HRD.dat`, `TRD.dat`, `BRD.dat`, `TRI.dat`, and/or `PSIGN.dat`, from the `SCALAPACK/TESTING` directory to your IBM SP-1 or SP-2 computer. As with the PB-BLAS tester, you must

then decide on the number of processors to be allocated for your testing/timing runs. In each input file, there is a set of processor grid dimensions specified (`VALUES OF P` and `VALUES OF Q`). The number of processors you allocate for an SP series machine must be greater than or equal to the maximum processor grid (`P` $\times$ `Q`) specified in the input file. The default input files assume 8 processors.

Then, for example, to run the tests of the DOUBLE PRECISION REAL LU routines, edit the `sp1.exe` file to load the executable `xdlu`. And type,

`sp1.exe` *[number of processors to allocate]*

omitting the square brackets.

Please note that the commands needed to load and run the executable are not standard across all SP series machines, the included script file `sp1.exe` runs the tests on an IBM SP-1 located at Cornell University.

Similar commands can be used for alternate precisions of the same test program or other test programs, where, `xdlu` is replaced by `xdls`, `xdqr`, `xdllt`, `xdhrd`, `xdtrd`, `xdbrd`, `xdtri`, `xdpsign`, or `xdsep`. The name of the output file is indicated on the third line of the input file and is currently defined to be `lu.out` for the LU tester, with similar names for the other data types. The user may also choose to send all output to standard error.

3.10 Send the Results to Tennessee

Congratulations! You have now finished installing and testing ScaLAPACK. Your participation is greatly appreciated. If possible, results and comments should be sent by electronic mail to

scalapack@cs.utk.edu

This first public release of ScaLAPACK is not compatible with any previous test release.

4 More About the ScaLAPACK Test Suite

The main test programs for the ScaLAPACK routines are located in the `SCALAPACK/TESTING/LIN` and `SCALAPACK/TESTING/EIG` subdirectories and are called `pd__driver.f` (`ps__driver.f` for REAL, `pc__driver.f` for COMPLEX, and `pz__driver.f` for COMPLEX*16), where the `__` is replaced by `lu`, `qr`, `llt`, and so on. Each of the test programs for the ScaLAPACK routines has a similar style of input.

Please note that only the SINGLE and DOUBLE PRECISION REAL symmetric eigenproblem driver is available at this time.

The following sections describe the different input formats and testing verifications. The data inside the input files is only test data designed to exercise the code. It should NOT be interpreted in any way as OPTIMAL performance values for any of the routines. For best performance using PVM, the largest possible blocksize NB should be used. Our experiments on the Intel machines suggest that a blocksize of NB equal to 6 is a good starting point.

The test programs for the routines are driven by separate data files from which the following types of parameters may be varied:

- TRANS, (*only used for LLS*)
- LOWER or UPPER, triangular input matrix (*ONLY for TRD*)
- M, the number of rows in the matrix (*not used for LLT, TRI, HRD, or TRD*)
- N, the order of the matrix (*not used for QR*)
- ILO and IHI (*ONLY for HRD*)
- NB, the blocksize for the blocked routines
- NRHS, the number of right hand sides (*only used for LU, LLT, and LLS*)
- NBRHS, the column blocksize for the rhs matrix B (*only used for LU, LLT, and LLS*)
- P, the grid row dimension
- Q, the grid column dimension
- THRESH, the acceptable threshold value for the residuals
- EST, logical flag to test cond. est. and it. ref. (*ONLY for LU and LLT*)

The number and size of the input values are limited by certain program maximums which are defined in PARAMETER statements in the main test programs. These program maximums are:

Parameter	Description	Value
TOTMEM	Total Memory available for testing data	6200000
INTGSZ	Length in bytes to store a INTEGER element	4
REALSZ	Length in bytes to store a REAL element	4
DBLESZ	Length in bytes to store a DOUBLE PRECISION element	8
CPLXSZ	Length in bytes to store a COMPLEX element	8
ZPLXSZ	Length in bytes to store a COMPLEX*16 element	16
NTESTS	Maximum number of tests to be performed	20

The user should modify TOTMEM to indicate the maximum amount of memory in bytes his system has available. You must remember to leave room in memory for the operating system, the BLACS buffer, etc. For example, on our system with 8 MB of memory, the parameters we use are TOTMEM=6,200,000 (leaving around 1.8 MB for OS, program, BLACS buffer, etc), and the length of a DOUBLE is 8. Some experimenting with the maximum allowable value of TOTMEM may be required. All arrays used by the factorizations, reductions, solves, and condition and error estimation are allocated out of the big array called MEM.

Please note that these parameter maximums in the test programs assume at least 8 Megabytes of memory per process. Thus, if you do not have that much space per process then you will need to reduce the size of the parameters.

For each of the test programs, the test program generates test matrices (nonsymmetric, symmetric, or symmetric positive-definite), calls the ScaLAPACK routines in that path, and computes a solve and/or factorization and/or reduction residual error check to verify that each operation has performed correctly. The factorization residual is only calculated if the residual for the solve step exceeds the threshold value THRESH. Thus, if a user wants both checks automatically done then he should set THRESH = 0.0.

When the tests are run, each test ratio that is greater than or equal to the threshold value causes a line of information to be printed to the output file.

A table of timing information is printed in the output file containing execution times as well as megaflop rates.

After all of the tests have been completed, summary lines are printed of the form

```
Finished  180 tests, with the following results:
  180 tests completed and passed residual checks.
    0 tests completed and failed residual checks.
    0 tests skipped because of illegal input values.

END OF TESTS.
```

4.1 Tests for the ScaLAPACK LU routines

The LU test program generates random nonsymmetric test matrices with values in the interval [-1,1], calls the ScaLAPACK routines to factor and solve the system, and computes a solve and/or factorization residual error check to verify that each operation has performed correctly. Condition estimation and iterative refinement routines are included and are optionally tested.

Specifically, each test matrix is subjected to the following tests:

- Factor the matrix $A = LU$ using PxGETRF
- Solve the system $AX = B$ using PxGETRS, and compute the ratio

  $$SRESID = ||AX - B||/(n||A||\,||X||\varepsilon)$$

- If $SRESID > THRESH$, then compute the ratio

  $$FRESID = ||LU - A||/(n||A||\varepsilon)$$

The expert driver (`PxGESVX`) performs condition estimation and iterative refinement and thus incorporates the following additional tests:

- Compute the reciprocal condition number RCOND using PxGECON. and compare to the value RCONDC which was computed as 1/(ANORM * AINVNM) where AINVNM is the explicitly computed norm of A^{-1}. The larger of the ratios

  RCOND/RCONDC and RCONDC/RCOND

  is returned. Since the same value of ANORM is used in both cases, this test measures the accuracy of the estimate computed for A^{-1}.

- Use iterative refinement (PxGERFS) to improve the solution, and recompute the ratio

$$SRESID = ||AX - B||/(n||A||\,||X||\varepsilon)$$

4.1.1 Input File for Testing the ScaLAPACK LU Routines

An annotated example of an input file for the test program is shown below.

```
'ScaLAPACK LU factorization input file'
'PVM machine.'
'lu.out'                output file name (if any)
6                       device out
2                       number of problems sizes
250 553                 values of N
3                       number of NB's
2 3 5                   values of NB
2                       number of NRHS's
1 5                     values of NRHS
3                       Number of NBRHS's
1 3 5                   values of NBRHS
5                       Number of processor grids (ordered pairs of P & Q)
1 4 2 1 8               values of P
1 2 4 8 1               values of Q
1.0                     threshold
T                       (T or F) Test Cond. Est. and Iter. Ref. Routines
```

4.2 Tests for the ScaLAPACK LLT routines

The Cholesky test program generates random symmetric test matrices with values in the interval [-1,1] and then modifies these matrices to be diagonally dominant with positive diagonal elements thus creating symmetric positive-definite matrices. It then calls the ScaLAPACK routines to factor and solve the system, and computes a solve and/or factorization residual error check to verify that each operation has performed correctly. Condition estimation and iterative refinement routines are included and optionally tested.

Specifically, each test matrix is subjected to the following tests:

- Compute the LLT factorization using PxPOTRF
- Solve the system $AX = B$ using PxPOTRS, and compute the ratio

$$SRESID = ||AX - B||/(n||A||\,||X||\varepsilon)$$

- IF $SRESID > THRESH$, then compute the ratio

$$FRESID = ||LL^T - A||/(n||A||\varepsilon)$$

The expert driver (`PxPOSVX`) performs condition estimation and iterative refinement and thus incorporates the following additional tests:

- Compute the reciprocal condition number RCOND using PxPOCON. and compare to the value RCONDC which was computed as 1/(ANORM * AINVNM) where AINVNM is the explicitly computed norm of A^{-1}. The larger of the ratios

  RCOND/RCONDC and RCONDC/RCOND

  is returned. Since the same value of ANORM is used in both cases, this test measures the accuracy of the estimate computed for A^{-1}.

- Use iterative refinement (PxPORFS) to improve the solution, and recompute the ratio

  $SRESID = ||AX - B||/(n||A||\,||X||\varepsilon)$

4.2.1 Input File for Testing the ScaLAPACK LLT Routines

An annotated example of an input file for the test program is shown below.

```
'ScaLAPACK LLT factorization input file'
'PVM machine.'
'lltest.out'                      output file name (if any)
6                                 device out
2                                 number of problems sizes
250 553                           values of N
3                                 number of NB's
2 3 5                             values of NB
2                                 number of NRHS's
1 5                               values of NRHS
3                                 Number of NBRHS's
1 3 5                             values of NBRHS
5                                 Number of processor grids (ordered pairs of P & Q)
1 4 2 8 1                         values of P
1 2 4 1 8                         values of Q
1.0                               threshold
T                                 (T or F) Test Cond. Est. and Iter. Ref. Routines
```

4.3 Tests for the ScaLAPACK QR, RQ, LQ, QL, and QP routines

The QR test program generates random nonsymmetric test matrices with values in the interval [-1,1], calls the ScaLAPACK routines to factor the system, and computes a factorization residual error check to verify that each operation has performed correctly.

Specifically, each test matrix is subjected to the following tests:

- Compute the QR factorization using PxGEQRF, and generate the orthogonal matrix Q from the Householder vectors

- Compute the ratio

  $FRESID = ||QR - A||/(n||A||\varepsilon)$

The testing of the RQ, LQ, QL, and QP routines proceeds in a similar fashion. Simply replace all occurrences of QR in the previous discussion with RQ, LQ, QL, or QP, respectively.

4.3.1 Input File for Testing the ScaLAPACK QR, RQ, LQ, QL, and QP Routines

An annotated example of an input file for the test program is shown below.

```
'ScaLAPACK QR factorizations input file'
'PVM machine'
'QR.out'                          output file name (if any)
6                                 device out
'QR' 'QL' 'LQ' 'RQ' 'QP'          factorization: QR, QL, LQ, RQ, QP
8                                 number of problems sizes
2 5 13 15 13 26 30 15             values of M
2 7 8  10 17 20 30 35             values of N
6                                 number of NB's
2 3 4 5 6 20                      values of NB
7                                 number of process grids (ordered pairs P & Q)
1 2 1 4 2 3 8                     values of P
1 2 4 1 3 2 1                     values of Q
3.0                               threshold
```

4.4 Tests for the Linear Least Squares (LLS) routines

The LLS test program solves the undetermined or overdetermined system of linear equations by generating random nonsymmetric test matrices with values in the interval [-1,1], calls the ScaLAPACK routines to solve the system, and computes a solve residual error check to verify that each operation has performed correctly.

Specifically, each test matrix is subjected to the following tests:

- If $M \geq N$, factor the matrix $A = QR$ using PxGEQRF
- Solve the system $AX = B$ using PxORMQR, and compute the ratio

  $$SRESID = ||AX - B||/(n||A||\,||X||\varepsilon)$$

- Else if $M < N$, factor the matrix $A = LQ$ using PxGELQF
- Solve the system $AX = B$ using PxORMLQ, and compute the ratio

  $$SRESID = ||AX - B||/(n||A||\,||X||\varepsilon)$$

4.4.1 Input File for Testing the ScaLAPACK LLS Routines

An annotated example of an input file for the test program is shown below.

```
'ScaLAPACK LLS input file'
'PVM machine.'
'lstest.out                     output file name (if any)
6                               device out
'N'                             TRANS ('N' or 'T')
2                               Number of problems sizes
250 553                         values of M
250 553                         values of N
3                               number of NB's
2 3 5                           values of NB
2                               number of NRHS's
1 5                             values of NRHS
3                               Number of NBRHS's
1 3 5                           values of NBRHS
5                               Number of processor grids (ordered pairs of P & Q)
1 4 2 1 8                       values of P
1 2 4 8 1                       values of Q
1.0                             threshold
```

4.5 Tests for the ScaLAPACK TRI routines

- Compute the LU factorization using PxGETRF, and then compute the inverse by invoking PxGETRI
- Compute the ratio

$$FRESID = ||AA^{-1} - I||/(n||A||\varepsilon)$$

4.5.1 Input File for Testing the ScaLAPACK TRI Routines

An annotated example of an input file for the test program is shown below.

```
'ScaLAPACK LU factorization + Inversion input file'
'CM5 32 nodes'
'TRI.out'                       output file name (if any)
6                               device out
8                               Number of problems sizes
2 5 10 15 13 20 30 35           values of N
6                               number of NB's
2 3 4 5 6 20                    values of NB
10                              Number of processor grids (ordered P & Q)
1 2 1 2 1 3 2 3 4 1             values of P
1 1 2 2 3 1 3 2 1 4             values of Q
1.0                             threshold
```

4.6 Tests for the ScaLAPACK HRD routines

The HRD test program generates random nonsymmetric test matrices with values in the interval [-1,1], calls the ScaLAPACK routines to reduce the test matrix to upper Hessenberg form, and computes a reduction residual error check to verify that each operation has performed correctly.

Specifically, each test matrix is subjected to the following tests:

- Reduce the matrix A to upper Hessenberg form H using PxGEHRD

  $$Q^T * A * Q = H.$$

- and compute the ratio

  $$FRESID = ||Q * H * Q^T - A||/(n||A||\varepsilon)$$

4.6.1 Input File for Testing the ScaLAPACK HRD Routines

An annotated example of an input file for the test program is shown below.

```
'ScaLAPACK HRD input file'
'PVM machine.'
'HRD.out'               output file name (if any)
6                       device out
1                       number of problems sizes
100 101                 values of N
1 1                     values of ILO
100 101                 values of IHI
1                       number of NB's
2 1 2 3 4 5             values of NB
1                       number of processor grids (ordered pairs of P & Q)
2 1 4                   values of P
2 4 1                   values of Q
1.0                     threshold
```

4.7 Tests for the ScaLAPACK TRD routines

The TRD test program generates random symmetric test matrices with values in the interval [-1,1], calls the ScaLAPACK routines to reduce the test matrix to symmetric tridiagonal form, and computes a reduction residual error check to verify that each operation has performed correctly.

Specifically, each test matrix is subjected to the following tests:

- Reduce the symmetric matrix A to symmetric tridiagonal form T using PxSYTRD

  $$Q^T * A * Q = T.$$

- and compute the ratio

  $$FRESID = ||Q * T * Q^T - A||/(n||A||\varepsilon)$$

4.7.1 Input File for Testing the SCALAPACK TRD Routines

An annotated example of an input file for the test program is shown below.

```
'ScaLAPACK TRD computation input file'
'PVM machine.'
'TRD.out'       output file name
6               device out
'L'             define Lower or Upper
2               number of problems sizes
16 17 100 101   values of N
3               number of NB's
3 4 5           values of NB
3               Number of processor grids (ordered pairs of P & Q)
2 4 1           values of P
2 1 4           values of Q
1.0             threshold
```

4.8 Tests for the ScaLAPACK BRD routines

The BRD test program generates random nonsymmetric test matrices with values in the interval [-1,1], calls the ScaLAPACK routines to reduce the test matrix to upper or lower bidiagonal form, and computes a reduction residual error check to verify that each operation has performed correctly.

Specifically, each test matrix is subjected to the following tests:

- Reduce the matrix A to upper or lower bidiagonal form B using PxGEBRD

  $$Q^T * A * P = B.$$

- and compute the ratio

  $$FRESID = ||Q * B * P^T - A||/(n||A||\varepsilon)$$

4.8.1 Input File for Testing the ScaLAPACK BRD Routines

An annotated example of an input file for the test program is shown below.

```
'ScaLAPACK BRD input file'
'PVM machine.'
'BRD.out'       output file name (if any)
6               device out
3               number of problems sizes
16 14 25 15 16  values of M
9  13 20 15 16  values of N
2               number of NB's
3 4 5           values of NB
3               Number of processor grids (ordered pairs of P & Q)
```

```
2 4 1           values of P
2 1 4           values of Q
1.0             threshold
```

4.9 Tests for the ScaLAPACK PSIGN routines

- Compute a low degree polynomial of A using PxGEPLN, then deflate the resulting matrix by calling PxGEDFL. The computation of the Sign function is performed by PxGESGN, PxGESGS or PxGESGH; the QR factorization with column pivoting by PxGEQPF; the post- and pre-multiplications by an orthogonal matrix by PxORMQR.
- If the matrix can be generated on only one node, compute the eigenvalues of the initial matrix and the deflated matrix using the LAPACK routine xGEEV. Then, compare the results by computing the maximum of the difference of the sorted eigenvalues. Otherwise no tests are performed. In a future release, this package will include a routine to compute eigenvalues and/or eigenvectors, and/or Schur vectors, so that it will be possible to perform checking tests for any matrix size.

4.9.1 Input File for Testing the ScaLAPACK PSIGN Routines

An annotated example of an input file for the DOUBLE PRECISION test program is shown below.

```
'ScaLAPACK Sign function deflation process input file'
'PVM machine.'
'PSIGN.out'                     output file name (if any)
6                               device out
1                               Number of problems sizes
10 200 300 400 500 600          values of N
1                               number of NB's
4                               values of NB
1                               Nb of processor grids (ordered P & Q)
2 4 8                           values of P
2 4 2                           values of Q
'R'                             Region of the plane: 'L', 'R', 'S', 'T', 'P'
1                               Nb of points to define the region
0.3D0 5.0D0 0.0D0 0.0D0         x1, x2, x3, x4 ...
'NABHR'                         Scaling Scheme: 'A', 'B', 'H', 'R', 'N'
'NSH'                           Iter. process: 'N', 'S', 'H'
1.0                             threshold
```

4.10 Tests for the ScaLAPACK SEP routines

The following tests will be performed on PDSYEVX:

$$r_1 \quad = \quad \frac{\|A - ZDZ^*\|}{n\, ulp\, \|A\|}$$

$$r_2 = \frac{\|I - ZZ^*\|}{n\ ulp}$$

$$r_3 = min_i \quad \frac{\|D1(i..i+m-1) - D2\|}{ulp\ \|D1\|}$$

where Z is the matrix of eigenvectors returned when the eigenvector option is given, $D1$ is the eigenvalues returned when the full eigendecomposition is requested. $D2$ is the eigenvalues returned when only eigenvalues are requested. m is the number of eigenvalues requested, and ulp represents xLAMCH('P').

The P_SYEVX tester allows multiple test requests to be controlled from a single input file. Each test request is controlled by the following inputs:

Values of N
N = The matrix size

Values of P, Q, NB
P = NPROW, the number of processor rows
Q = NPCOL, the number of processor columns
NB = the block size

Values of the matrix types
See Section 4.10.1.

Number of eigen requests
1 = Test full eigendecomposition only
8 = Test the following eigen requests:
Full eigendecomposition
All eigenvalues, no eigenvectors
Eigenvalues requested by value (i.e. VL,VU)
Eigenvalues and vectors requested by value
Eigenvalues requested by index (i.e. IL, IU)
Eigenvalues and vectors requested by index
Full eigendecomposition with minimal workspace provided
Full eigendecomposition with random workspace provided

Threshold
The highest value of r_1, r_2 and r_3 that will be accepted.

Absolute tolerance
Must be -1.0 to ensure orthogonal eigenvectors

Print Request
1 = Print every test
2 = Print only failing tests and a summary of the request

4.10.1 Test Matrices for the Symmetric Eigenvalue Routines

Twenty-two different types of test matrices may be generated for the symmetric eigenvalue routines. Table 1 shows the types, along with the numbers used to refer to the matrix types. Except as noted, all matrices have norm $O(1)$. The expression UDU^{-1} means a real diagonal matrix D with entries of magnitude $O(1)$ conjugated by a unitary (or real orthogonal) matrix U.

	Eigenvalue Distribution			
Type	Arithmetic	Geometric	Clustered	Other
Zero				1
Identity				2
Diagonal	3	$4, 6^{\dagger}, 7^{\ddagger}$	5	
UDU^{-1}	$8, 11^{\dagger}, 12^{\ddagger}, 16^{*}, 19^{\star}, 20^{\bullet}$	$9, 17^{*}$	$10, 18^{*}$	
Symmetric w/Random entries				$13, 14^{\dagger}, 15^{\ddagger}$
Tridiagonal				21^{a}
Multiple Clusters				22^{b}

$\dagger$ – matrix entries are $O(\sqrt{\text{overflow}})$
$\ddagger$ – matrix entries are $O(\sqrt{\text{underflow}})$
$*$ – diagonal entries are positive
$\star$ – matrix entries are $O(\sqrt{\text{overflow}})$ and diagonal entries are positive
$\bullet$ – matrix entries are $O(\sqrt{\text{underflow}})$ and diagonal entries are positive
a – Some of the immediately off-diagonal elements are zero - guaranteeing splitting
b – Clusters are sized: $1, 2, 4, \ldots, 2^{i}$.

Table 1: Test matrices for the symmetric eigenvalue problem

4.10.2 Input File for Testing the Symmetric Eigenvalue Routines and Drivers

An annotated example of an input file for testing the symmetric eigenvalue routines and drivers is shown below.

```
'ScaLAPACK Symmetric Eigensolver input file'
'PVM Machine'
'sep.out'                       output file name (not supported)
6                               device out
4                               maximum number of processes
' '
'TEST 1 - test small matrices - '
5                               number of matrices
0 1 2 3 4
2                               number of uplo choices
'L' 'U'                         uplo choices
2                               number of processor configurations (P, Q, NB)
```

```
2 1                             values of P (NPROW)
1 2                             values of Q (NPCOL)
1 2                             values of NB
2                               number of matrix types
8 9                             matrix types
8                               number of eigenvalue request types (1 or 8)
10.0                            Threshold
-1                              Absolute Tolerance
2                               print request (2= summary print only)
' '
'TEST 2 - test small matrices - all processor configurations'
3                               number of matrices
2 3 4
2                               number of uplo choices
'L' 'U'                         uplo choices
13                              number of processor configurations (P, Q, NB)
1 1 2 1 2 1 3 1 3 1 2 2 2       values of P (NPROW)
1 1 1 2 1 2 1 3 1 3 2 2 2       values of Q (NPCOL)
1 3 1 1 2 2 1 1 2 2 1 2 3       values of NB
1                               number of matrix types
8                               matrix types
1                               number of eigenvalue request types (1 or 8)
10.0                            Threshold
-1                              Absolute Tolerance
2                               print request (2= summary print only)
' '
'TEST 3 - test medium matrices - all types and requests'
2                               number of matrices
21 63
1                               number of uplo choices
'U'                             uplo choices
1                               number of processor configurations (P, Q, NB)
2                               values of P (NPROW)
2                               values of Q (NPCOL)
8                               values of NB
22                              number of matrix types
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 matrix types
1                               number of eigenvalue request types (1 or 8)
10.0                            Threshold
-1                              Absolute Tolerance
1                               print request (2= summary print only)
' '
'TEST 4 - test medium matrices - all processor configurations'
2                               number of matrices
25 36
```

```
2                               number of uplo choices
'U' 'L'                         uplo choices
13                              number of processor configurations (P, Q, NB)
1 1 2 1 2 1 3 1 3 1 2 2 2       values of P (NPROW)
1 1 1 2 1 2 1 3 1 3 2 2 2       values of Q (NPCOL)
1 3 1 1 2 2 1 1 2 2 1 2 3       values of NB
1                               number of matrix types
8                               matrix types
1                               number of eigenvalue request types (1 or 8)
20.0                            Threshold
-1                              Absolute Tolerance
1                               print request (2= summary print only)
' '
'TEST 5 - test medium matrices - a bit of everything'
2                               number of matrices
28 39
2                               number of uplo choices
'L' 'U'                         uplo choices
3                               number of processor configurations (P, Q, NB)
2 1 2                           values of P (NPROW)
2 2 1                           values of Q (NPCOL)
8 3 13                          values of NB
2                               number of matrix types
8 9                             matrix types
8                               number of eigenvalue request types (1 or 8)
10.0                            Threshold
-1                              Absolute Tolerance
1                               print request (2= summary print only)
' '
'TEST 6 - test one large matrix'
1                               number of matrices
600
1                               number of uplo choices
'L'                             uplo choices
1                               number of processor configurations (P, Q, NB)
2                               values of P (NPROW)
2                               values of Q (NPCOL)
8                               values of NB
1                               number of matrix types
8                               matrix types
1                               number of eigenvalue request types (1 or 8)
4.0                             Threshold
-1                              Absolute Tolerance
1                               print request (2= summary print only)
' '
```

```
'End of tests'
-1
```

Appendix A

ScaLAPACK Routines

In this appendix, we review the subroutine naming scheme for ScaLAPACK and indicate by means of a table which subroutines are included in this release. We also list the driver routines.

Each subroutine name in ScaLAPACK, which has an LAPACK equivalent, is simply the LAPACK name prepended by a `P`. All names consist of seven characters in the form PTXXYYY. The second letter, T, indicates the matrix data type as follows:

S	REAL
D	DOUBLE PRECISION
C	COMPLEX
Z	COMPLEX*16 (if available)

The next two letters, XX, indicate the type of matrix. Most of these two-letter codes apply to both real and complex routines; a few apply specifically to one or the other, as indicated below:

GE	general (i.e. unsymmetric, in some cases rectangular)
HE	(complex) Hermitian
OR	(real) orthogonal
PO	symmetric or Hermitian positive definite
ST	symmetric tridiagonal
SY	symmetric
TR	triangular (or in some cases quasi-triangular)
UN	(complex) unitary

The last three characters, YYY, indicate the computation done by a particular subroutine. Included in this release are subroutines to perform the following computations:

BRD	reduce to bidiagonal form by orthogonal transformations
CON	estimate condition number
EBZ	compute selected eigenvalues by bisection
EIN	compute selected eigenvectors by inverse iteration
EQR	compute eigenvalues and/or the Schur form using the QR algorithm

EQU	equilibrate a matrix to reduce its condition number
GBR	generate the orthogonal/unitary matrix from PxGEBRD
GHR	generate the orthogonal/unitary matrix from PxGEHRD
GLQ	generate the orthogonal/unitary matrix from PxGELQF
GQL	generate the orthogonal/unitary matrix from PxGEQLF
GQR	generate the orthogonal/unitary matrix from PxGEQRF
GRQ	generate the orthogonal/unitary matrix from PxGERQF
GTR	generate the orthogonal/unitary matrix from PxxxTRD
HRD	reduce to upper Hessenberg form by orthogonal transformations
LQF	compute an LQ factorization without pivoting
MBR	multiply by the orthogonal/unitary matrix from PxGEBRD
MHR	multiply by the orthogonal/unitary matrix from PxGEHRD
MLQ	multiply by the orthogonal/unitary matrix from PxGELQF
MQL	multiply by the orthogonal/unitary matrix from PxGEQLF
MQR	multiply by the orthogonal/unitary matrix from PxGEQRF
MRQ	multiply by the orthogonal/unitary matrix from PxGERQF
MTR	multiply by the orthogonal/unitary matrix from PxxxTRD
QLF	compute a QL factorization without pivoting
QPF	compute a QR factorization with column pivoting
QRF	compute a QR factorization without pivoting
RFS	refine initial solution returned by TRS routines
RQF	compute an RQ factorization without pivoting
TRD	reduce a symmetric matrix to real symmetric tridiagonal form
TRF	compute a triangular factorization (LU, Cholesky, etc.)
TRI	compute inverse (based on triangular factorization)
TRS	solve systems of linear equations (based on triangular factorization)

Given these definitions, the following table indicates the ScaLAPACK subroutines for the solution of systems of linear equations:

	GE	GG	GB	GT	PO	PP	PB	PT	HE SY	HP SP	TR	TP	TB	UN OR
TRF	×				×									
TRS	×				×						×			
RFS	×				×									
TRI	×				×						×			
CON	×				×									
EQU	×				×									
QPF	×													
QRF†	×													
GQR†														×
MQR†														×

†– also RQ, QL, and LQ

The following table indicates the ScaLAPACK subroutines for finding eigenvalues and eigenvectors or singular values and singular vectors:

	GE	GB	GG	HS	HG	TR	TG	HE SY	HP SP	HB SB	ST	PT	BD
HRD	×												
TRD								×					
BRD	×												
EQR											×		
EQZ													
EIN											×		
EBZ											×		

Orthogonal/unitary transformation routines have also been provided for the reductions that use elementary transformations.

	UN OR	UP OP
GHR	×	
GTR	×	
GBR	×	
MHR	×	
MTR	×	
MBR	×	

In addition, a number of driver routines are provided with this release. The naming convention for the driver routines is the same as for the LAPACK routines, but the last 3 characters YYY have the following meanings (note an 'X' in the last character position indicates a more expert driver):

SV	factor the matrix and solve a system of equations
SVX	equilibrate, factor, solve, compute error bounds and do iterative refinement, and estimate the condition number
LS	solve over- or underdetermined linear system using orthogonal factorizations
EV	compute all eigenvalues and/or eigenvectors
EVX	compute selected eigenvalues and eigenvectors

The driver routines provided in ScaLAPACK are indicated by the following table:

	GE	GG	GB	GT	PO	PP	PB	PT	HE SY	HP SP	HB SB	ST
SV	×				×							
SVX	×				×							
LS	×											
EV												
EVX									×			

Appendix B

ScaLAPACK Auxiliary Routines

This appendix lists all of the auxiliary routines (except for the BLAS and LAPACK) that are called from the ScaLAPACK routines. These routines are found in the directory `SCALAPACK/SRC`. Routines specified with a first character P followed by an underscore as the second character are available in all four data types (S, D, C, and Z), except those marked (real), for which the first character may be 'S' or 'D', and those marked (complex), for which the first character may be 'C' or 'Z'.
Functions for computing norms:

P_LANGE General matrix
P_LANHE (complex) Hermitian matrix
P_LANHS Upper Hessenberg matrix
P_LANSY Symmetric matrix
P_LANTR Trapezoidal matrix

Level 2 BLAS versions of the block routines:

P_GEBD2 reduce a general matrix to bidiagonal form
P_GEHD2 reduce a square matrix to upper Hessenberg form
P_GELQ2 compute an LQ factorization without pivoting
P_GEQL2 compute a QL factorization without pivoting
P_GEQR2 compute a QR factorization without pivoting
P_GERQ2 compute an RQ factorization without pivoting
P_GETF2 compute the LU factorization of a general matrix
P_HETD2 (complex) reduce a Hermitian matrix to real tridiagonal form
P_ORG2L (real) generate the orthogonal matrix from PxGEQLF
P_ORG2R (real) generate the orthogonal matrix from PxGEQRF
P_ORGL2 (real) generate the orthogonal matrix from PxGEQLF
P_ORGR2 (real) generate the orthogonal matrix from PxGERQF
P_ORM2L (real) multiply by the orthogonal matrix from PxGEQLF
P_ORM2R (real) multiply by the orthogonal matrix from PxGEQRF
P_ORMBR (real) multiply by the orthogonal matrix from PxGEBRD
P_ORMHR (real) multiply by the orthogonal matrix from PxGEHRD
P_ORML2 (real) multiply by the orthogonal matrix from PxGELQF

P_ORMR2 (real) multiply by the orthogonal matrix from PxGERQF
P_ORMTR (real) multiply by the orthogonal matrix from PxSYTRD
P_POTF2 compute the Cholesky factorization of a positive definite matrix
P_SYTD2 (real) reduce a symmetric matrix to tridiagonal form
P_TRTI2 compute the inverse of a triangular matrix
P_UNG2L (complex) generate the unitary matrix from PxGEQLF
P_UNG2R (complex) generate the unitary matrix from PxGEQRF
P_UNGL2 (complex) generate the unitary matrix from PxGEQLF
P_UNGR2 (complex) generate the unitary matrix from PxGERQF
P_UNM2L (complex) multiply by the unitary matrix from PxGEQLF
P_UNM2R (complex) multiply by the unitary matrix from PxGEQRF
P_UNML2 (complex) multiply by the unitary matrix from PxGELQF
P_UNMR2 (complex) multiply by the unitary matrix from PxGERQF

Other ScaLAPACK auxiliary routines:

P_LABAD	(real) returns square root of underflow and overflow if exponent range is large
P_LABRD	reduce NB rows or columns of a matrix to upper or lower bidiagonal form
P_LACGV	(complex) conjugates a complex vector of length n
PDLACHKIEEE	performs a simple check for the features of the IEEE standard
P_LACON	estimate the norm of a matrix for use in condition estimation
P_LACPY	copy a matrix to another matrix
PDLAEVSWP	moves the eigenvectors from where they are computed to a standard block cyclic array
P_LAHRD	reduce NB columns of a general matrix to Hessenberg form
PDLAIECTB	computes the number of negative eigenvalues in $(A - \Sigma I)$ where the sign bit is assumed to be bit 32.
PDLAIECTL	computes the number of negative eigenvalues in $(A - \Sigma I)$ where the sign bit is assumed to be bit 64.
P_LAQGE	equilibrate a general matrix
P_LAQSY	equilibrate a symmetric matrix
PDLARED1D	redistributes a one-dimensional array
P_LARF	apply (multiply by) an elementary reflector
P_LARFB	apply (multiply by) a block reflector
P_LARFG	generate an elementary reflector
P_LARFT	form the triangular factor of a block reflector
P_LASCL	scale a matrix by CTO/CFROM
P_LASET	initializes a matrix to BETA on the diagonal and ALPHA on the off-diagonals
PDLASNBT	computes the position of the sign bit of a double precision floating point number
P_LASSQ	Compute a scaled sum of squares of the elements of a vector
P_LASWP	Perform a series of row interchanges
P_LATRD	reduce NB rows and columns of a real symmetric or complex Hermitian matrix to tridiagonal form
P_LATRS	solve a triangular system with scaling to prevent overflow

P_LAUU2	Unblocked version of P_LAUUM
P_LAUUM	Compute the product U*U' or L'*L (blocked version)

Bibliography

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK Users' Guide*, Second Edition, SIAM, Philadelphia, PA, 1995.

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel. J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, *LAPACK: A Portable Linear Algebra Library for High-Performance Computers*, University of Tennessee, CS-90-105, May 1990.

[3] E. Anderson, A. Benzoni, J. Dongarra, S. Moulton, S. Ostrouchov, B. Tourancheau, and R. van de Geijn, *Basic Linear Algebra Communication Subprograms*. Sixth Distributed Memory Computing Conference Proceedings, IEEE Computer Society Press, 1991.

[4] E. Anderson, J. Dongarra, and S. Ostrouchov, *LAPACK Working Note 41: Installation Guide for LAPACK*, University of Tennessee, CS-92-151, February 1992.

[5] Z. Bai and J. Demmel, *Design of a Parallel Nonsymmetric Eigenroutine Toolbox*, Computer Science Tech. Report UCB/CSD-92-718, U.C.Berkeley, 1992.

[6] C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, and D. Sorensen, *LAPACK Working Note #5: Provisional Contents*, Argonne National Laboratory, ANL-88-38, September 1988.

[7] J. Choi, J. J. Dongarra, R. Pozo, and D. W. Walker, *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, 1992.

[8] J. Choi, J. J. Dongarra, and D. W. Walker, *PUMMA: Parallel Universal Matrix Multiplication Algorithms*, Technical Report ORNL/TM-12252, Oak Ridge National Laboratory, Mathematical Sciences Section, Oak Ridge, Tennessee, August 1993.

[9] J. Choi, J. J. Dongarra, and D. W. Walker, *Parallel Matrix Transpose Algorithms on Distributed Memory Concurrent Computers*, Technical Report ORNL/TM-12309, Oak Ridge National Laboratory, Mathematical Sciences Section, Oak Ridge, Tennessee, October 1993.

[10] J. Choi, J. J. Dongarra, and D. W. Walker, *PUMMA Reference Manual*, Technical Report ORNL/TM-12494, Oak Ridge National Laboratory, Mathematical Sciences Section, Oak Ridge, Tennessee, (in preparation) 1993.

[11] J. Choi, J. J. Dongarra, S. Ostrouchov, A. P. Petitet, D. W. Walker, R. C. Whaley, *The Design and Implementation of the ScaLAPACK LU, QR, and Cholesky Factorization Routines*, LAPACK Working Note 80, Technical Report CS-94-246, University of Tennessee, September, 1994.

[12] J. Choi, J. J. Dongarra, S. Ostrouchov, A. Petitet, and R. C. Whaley, *A Proposal for a Set of Parallel Basic Linear Algebra Subprograms*, In preparation, 1995.

[13] J. Demmel and K. Stanley, *The Performance of Finding Eigenvalues and Eigenvectors of Dense Symmetric Matrices on Distributed Memory Computers*, LAPACK Working Note 86, Technical Report CS-94-254, September, 1994.

[14] J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling, "A Set of Level 3 Basic Linear Algebra Subprograms," *ACM Trans. Math. Soft.*, 16, 1:1-17, March 1990.

[15] J. Dongarra, J. Du Croz, S. Hammarling, and R. Hanson, "An Extended Set of Fortran Basic Linear Algebra Subprograms," *ACM Trans. Math. Soft.*, 14, 1:1-17, March 1988.

[16] J. Dongarra and R. van de Geijn, *Two Dimensional Basic Linear Algebra Communication Subprograms*, LAPACK Working Note 37, technical report, University of Tennessee, 1991.

[17] G. A. Geist, A. L. Beguelin, J. J. Dongarra, W. Jiang, R. J. Manchek, and V. S. Sunderam., *PVM 3.3 User's Guide and Reference Manual*, Technical Report ORNL/TM-12187, Oak Ridge National Laboratory, Oak Ridge, Tennessee, September 1994.

[18] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, "Basic Linear Algebra Subprograms for Fortran Usage," *ACM Trans. Math. Soft.*, 5, 3:308-323, September 1979.

[19] R. Clint Whaley, *Basic Linear Algebra Communication Subprograms: Analysis and Implementation Across Multiple Parallel Architectures*, LAPACK Worling Note 73, Technical Report CS-94-234, University of Tennessee, May 1994.