LAPACK Working Note 93
Installation Guide for ScaLAPACK[1]

L. S. Blackford, A. Cleary[2], J. Choi[3],
J. J. Dongarra, A. Petitet[4], and R. C. Whaley
Department of Computer Science
University of Tennessee
Knoxville, Tennessee 37996-3450

and

J. Demmel, I. Dhillon[5], O. Marques[6], and K. Stanley
Computer Science Division
University of California, Berkeley
Berkeley, CA 94720

and

D. Walker[7]

VERSION 1.7: August 31, 2001

Abstract

This working note describes how to install and test version 1.7 of ScaLAPACK. This release of ScaLAPACK extends the functionality of the package by providing new routines for the nonsymmetric eigenproblem, divide and conquer symmetric eigenproblem, and improved routines for generalized symmetric eigenproblem and the reduction to tridiagonal form. It also includes PBLAS v2.0. 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.

[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 DAAH04-95-1-0077, 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.
[2]Current address: LLNL
[3]Current address: Soongsil University, Seoul, Korea
[4]Current address: Sun France, Paris, France
[5]Current address: IBM Austin
[6]Current address: LBL
[7]Current address: Cardiff University, Wales

Contents

1 Introduction

This working note describes how to install and test version 1.7 of ScaLAPACK [1]. This release of ScaLAPACK extends the functionality of the package by providing new routines for the nonsymmetric eigenproblem, divide and conquer symmetric eigenproblem, and improved routines for generalized symmetric eigenproblem and the reduction to tridiagonal form. It also includes PBLAS v2.0. For a detailed explanation of the design and contents of the ScaLAPACK library, please refer to the ScaLAPACK Users' Guide[1].

ScaLAPACK is freely available on *netlib* and can be obtained via the World Wide Web or anonymous ftp.

`http://www.netlib.org/scalapack/scalapack.tgz`

Prebuilt ScaLAPACK libraries are available on *netlib* for a variety of architectures.

`http://www.netlib.org/scalapack/archives/`

However, if a prebuilt library does not exist for your architecture, you will need to download the distribution tar file and build the library as instructed in this guide.

To install and test ScaLAPACK, the user must have the BLACS, BLAS[9, 6, 5], and MPI [7] or PVM [8] available on his machine.

ScaLAPACK has been tested on MPPs like the IBM SP series, Cray T3E, and SGI Origin 2000/3000, and tested on clusters of PCs and networks of workstations supporting MPI or PVM.[8]

Section 2 contains step-by-step installation and testing/timing instructions. For users desiring additional information, Section 3 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 Installation Procedure

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

1. Gunzip and tar the file `scalapack.tgz`.
2. Edit the `SLmake.inc` include file, specifying the location of the MPI or PVM library, the BLACS library, and the BLAS library.
3. Edit the top-level `Makefile`, and type `make`.
4. Run the Test Suite(s).

If failures are encountered during any phase of the installation or testing process, please first refer to the FAQ and Errata files for information

[8]It is very important to note that only PVM version 3.3 or later is supported with the BLACS[4, 10]. Due to major changes in PVM and the resulting changes required in the BLACS, earlier versions of PVM are NOT supported.

```
http://www.netlib.org/scalapack/faq.html
```

```
http://www.netlib.org/scalapack/errata.html
```

and if that does not resolve the problem, please contact the developers at

```
scalapack@cs.utk.edu
```

2.1 Gunzip and tar the file `scalapack.tgz`

The software is distributed in the form of a gzipped tar file which contains the ScaLAPACK source code and test suite, as well as the PBLAS source code and testing/timing programs. The PBLAS are parallel versions of the Level 1, 2, and 3 BLAS. For more details on the PBLAS, refer to [2, 3].

```
http://www.netlib.org/scalapack/scalapack.tgz
```

To unpack the `scalapack.tgz` file, type the following command:

```
gunzip -c scalapack.tgz | tar xvf -
```

This will create a top-level directory called `SCALAPACK` 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. Libraries are created in the

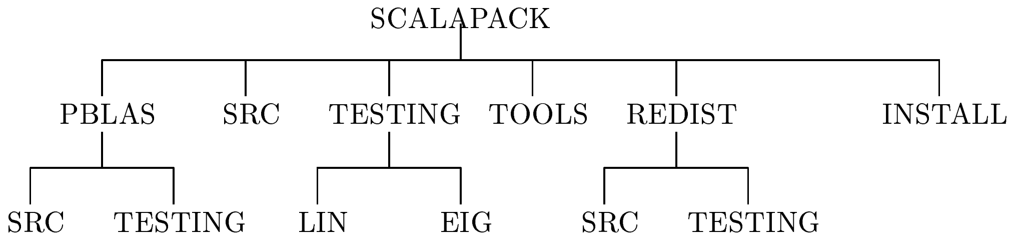

Figure 1: Organization of ScaLAPACK

`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. You will need approximately 28 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.

2.2 Edit the `SLmake.inc` include file

Example machine-specific `SCALAPACK/SLmake.inc` files are provided in the `INSTALL` subdirectory for the Intel i860, IBM SP, Cray T3E, SGI Origin, and various workstations using MPI or PVM. 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 INSTALL/SLmake.ALPHA SLmake.inc
```

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 require 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`, `PVM` or `MPI`, and `BLAS`.

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.

2.2.1 Further Details to obtain BLACS, BLAS, and PVM or MPI

Prebuilt BLACS libraries are available on *netlib* for a variety of architectures and message passing library combinations;

```
http://www.netlib.org/blacs/archives
```

otherwise, the BLACS distribution tar files are available.

```
http://www.netlib.org/blacs/mpiblacs.tgz
http://www.netlib.org/blacs/pvmblacs.tgz
```

After obtaining the source, follow the instructions in "A User's Guide to the BLACS" or in the "Installing the BLACS" section of the BLACS webpage 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 *blacs index* on *netlib*.

If an vendor optimized BLAS library is not available, then the user can install ATLAS which will generate an optimized BLAS library for the given architecture, or install the Fortran77 reference implementation of the BLAS.

```
http://www.netlib.org/blas/faq.html#1.6
http://www.netlib.org/atlas/
http://www.netlib.org/blas/blas.tgz
```

An optimized BLAS library is essential for best performance, and use of the Fortran77 reference implementation BLAS is strongly discouraged.

If a vendor-supplied MPI or PVM library is not available, portable implementations of PVM and MPI (MPICH and LAM/MPI) are available:

```
http://www.netlib.org/pvm3/
http://www.mcs.anl.gov/mpi/mpich/
http://www.lam-mpi.org/
```

Installation instructions for PVM are contained in the PVM Users' Guide [8]. An Installation Guide for MPICH is available on the aforementioned webpage. Likewise, installation instructions for LAM/MPI are contained on its respective webpage.

2.3 Edit the top-level `SCALAPACK/Makefile` and type `make`

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 the ScaLAPACK library are:

```
cd SCALAPACK

make
```

If you wish to build the testing executables (assuming that all libraries have previously been built), you can specify

`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 2.4 for details on the PBLAS Test Suite, section 2.6 to run the REDIST test suite, and section 2.7 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 testing routine object files and executable files.

2.4 Run the PBLAS Test Suite

The PBLAS testing executables are created in the `PBLASTSTdir` directory as defined in `SLmake.inc`. By default, these testing executables are copied into the `SCALAPACK/TESTING` directory. For the Level 1 PBLAS routines, the testing executables are called `xspblas1tst`, `xdpblas1tst`, `xcpblas1tst`, and `xzpblas1tst`. Likewise, the testing executables for the Level 2 PBLAS are `xspblas2tst`, `xdpblas2tst`, `xcpblas2tst`, and `xzpblas2tst`. The testing executables for the Level 3 PBLAS are `xspblas3tst`, `xdpblas3tst`, `xcpblas3tst`, and `xzpblas3tst`. There is one input file associated with each testing executable. For example, the input file for `xspblas1tst` is called `PSBLA1TST.dat`. The input files are copied to the `PBLASTSTdir` directory at the time the executables are built.

For brevity, we shall only list instructions for testing PBLAS executables using MPICH on a network of workstations, and PVM on a network of workstations. Execution instructions for the various distributed-memory computers are machine-dependent.

Testing instructions with MPICH on a network of workstations

For the sake of an example, we shall assume that you have installed the portable implementation of MPI, called `MPICH`, and built the PBLAS tester executables for each of the machines used in your application. The executable files are not required to be stored in a particular directory. Then, to run the executable, you will use the command `mpirun`. For example,

```
mpirun -np <number of processes> <executable>
```

where `<executable>` is replaced by `xspblas1tst`, and so on. If the network of workstations is heterogeneous, you will need to specify the `-p4pg` option and supply a text file containing the names of the machines and the locations of the executables to which you will spawn tasks. Refer to the `mpirun` manpage for complete details.

Testing instructions with PVM on a network of workstations

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 1 PBLAS, start the test program by typing:

```
xspblas1tst
```

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 `PSBLA1TST.dat` is located in the same directory as `xspblas1tst`. It is read on the machine from which you type `xspblas1tst` 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 `PSBLA1TST.SUMM` for the REAL version, with similar names for the other data types. The user may also choose to send all output to standard error.

2.5 Run the PBLAS Timing Suite (optional)

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

b) Type `make` followed by the data types desired. For the Level 1 PBLAS routines, the timing executables are called `xspblas1tim`, `xdpblas1tim`, `xcpblas1tim`, and `xzpblas1tim`, and are created in the `PBLASTSTdir` directory as defined in `SLmake.inc`. Likewise, the timing executables for the Level 2 PBLAS are `xspblas2tim`, `xdpblas2tim`, `xcpblas2tim`, and `xzpblas2tim`. The timing executables for the Level 3 PBLAS are `xspblas3tim`, `xdpblas3tim`, `xcpblas3tim`, and `xzpblas3tim`. There is one input file associated with each timing executable. For example, the input file for `xspblas1tim` is called `PSBLA1TIM.dat`. The input files are copied to the `PBLASTSTdir` directory at the time the executables are built.

c) Run the timing executables on the desired platform as analogously described in Section 2.4.

2.6 Run the REDIST Test Suite

The redistribution/copy routines allow the redistribution of a 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.

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`, and `xztrmr` 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.

2.7 Run the ScaLAPACK Test Suite

There are eighteen distinct test programs for testing the ScaLAPACK routines of the following type: LU, Cholesky, Band LU, Band Cholesky, General Tridiagonal, Band Tridiagonal, QR (RQ, LQ, QL, QP, and TZ), Linear Least Squares, upper Hessenberg reduction, tridiagonal reduction, bidiagonal reduction, matrix inversion, the symmetric eigenproblem, the generalized symmetric eigenproblem, the nonsymmetric eigenproblem, and the singular value decomposition.

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 singular value decomposition 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 3.

Run the testing executables on the desired platform as analogously described in Section 2.4. For example, in double precision, the testing executables are named xdlu, xdllt, xddblu, xdgblu, xddtlu, xdpbllt, xdptllt, xdls, xdqr, xdhrd, xdtrd, xdbrd, xdinv, xdsep, xdgsep, xdnep, and xdsvd. The input files are LU.dat, LLT.dat, BLU.dat, BLLT.dat, LS.dat, QR.dat, HRD.dat, TRD.dat, BRD.dat, INV.dat, SEP.dat, NEP.dat, and SVD.dat.

Similar commands can be used for alternate precisions of the same test program or other test programs. 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.

2.8 Troubleshooting

If failures are encountered during any phase of the installation or testing process, please first refer to the FAQ and Errata files for information

http://www.netlib.org/scalapack/faq.html

http://www.netlib.org/scalapack/errata.html

and if that does not resolve the problem, please contact the developers at

scalapack@cs.utk.edu

This release of ScaLAPACK is compatible with the previous release (version 1.6).

3 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.

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, the value of the blocksize NB should be set to the value determined by ATLAS as optimal. A good starting point is a multiple of 16 – e.g., 16, 32, 48, 64.

The test programs for the routines are driven by separate data files.

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	2000000
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, for PVM, the parameters we use are TOTMEM=2,000,000, 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 2 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, symmetric positive-definite, or upper Hessenberg), 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.
```

3.1 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 test:

- Compute the reciprocal condition number RCOND using PxGECON.
- Use iterative refinement (PxGERFS) to improve the solution, and recompute the ratio

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

3.1.1 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'
'MPI 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
```

3.2 Tests for the ScaLAPACK Band and Tridiagonal LU routines

The LU test program generates random nonsymmetric band 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.

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

- Compute the Band or Tridiagonal LU factorization using PxDBTRF (PxGBTRF or PxDTTRF)
- Solve the system $AX = B$ using PxDBTRS (PxGBTRS or PxDTTRS), and compute the ratio

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

3.2.1 Input File for Testing the ScaLAPACK Band and Tridiagonal LU Routines

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

```
'ScaLAPACK, Version 1.5, banded linear systems input file'
'PVM.'
''                              output file name (if any)
6                               device out
'T'                             define transpose or not
 7 3 4 8                                number of problem sizes
 2 5 17 28 37 121 200 1023 2048 3073    values of N
 6                               number of bandwidths
 1 2 3 15 6  8             values of BWL
 2 1 1 4  15 6              values of BWU
1                               number of NB's
-1 3 4 5                        values of NB (-1 for automatic determination)
1                               number of NRHS's (must be 1)
8                               values of NRHS
1                               number of NBRHS's (ignored)
1                               values of NBRHS (ignored)
4                              number of process grids
1 2 3 4 5 7 8 15 26 47 64      values of "Number of Process Columns"
3.0                             threshold
```

3.3 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.
- Use iterative refinement (PxPORFS) to improve the solution, and recompute the ratio

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

3.3.1 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'
'MPI 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
```

3.4 Tests for the ScaLAPACK Band and Tridiagonal LLT routines

The Cholesky test program generates random symmetric positive definite band or tridiagonal test matrices with values in the interval [-1,1]. It then calls the ScaLAPACK routines to factor and 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:

- Compute the Band or Tridiagonal LLT factorization using PxPBTRF (or PxPTTRF)
- Solve the system $AX = B$ using PxPBTRS (or PxPTTRS), and compute the ratio

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

3.4.1 Input File for Testing the ScaLAPACK Band or Tridiagonal LLT Routines

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

```
'ScaLAPACK, banded linear systems input file'
'PVM.'
''                              output file name (if any)
6                               device out
'L'                             define Lower or Upper
7                               number of problem sizes
1 5 17 28 37 121 200            values of N
6                               number of bandwidths
1 2 4 10 31 64                  values of BW
1                               number of NB's
-1 3 4 5                        values of NB (-1 for automatic determination)
1                               number of NRHS's (must be 1)
8                               values of NRHS
1                               number of NBRHS's (ignored)
1                               values of NBRHS (ignored)
4                               number of process grids
1 2 3 4 5 7                     values of "Number of Process Columns"
3.0                             threshold
```

3.5 Tests for the ScaLAPACK QR, RQ, LQ, QL, QP, and TZ 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. For TZ, the factorization routine is called PxTZRZF.

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

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

```
'ScaLAPACK, Orthogonal factorizations input file'
'MPI machine'
'QR.out'                         output file name (if any)
6                                device out
6                                number of factorizations
'QR' 'QL' 'LQ' 'RQ' 'QP' 'TZ'    factorizations: QR, QL, LQ, RQ, QP, TZ
4                                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
4                                number of blocking sizes
4 3 5 5 4 6                      values of MB
4 7 3 5 8 2                      values of NB
4                                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
```

3.6 Tests for the Linear Least Squares (LLS) routines

The LLS test program tests the PxGELS driver routine for computing solutions to over- and underdetermined, full-rank systems of linear equations $AX = B$ (A is m-by-n). For each test matrix type, we generate three matrices: One which is scaled near underflow, a matrix with moderate norm, and one which is scaled near overflow.

The PxGELS driver computes the least-squares solutions (when $m \geq n$) and the minimum-norm solution (when $m < n$) for an m-by-n matrix A of full rank. To test PxGELS, we generate a diagonally dominant matrix A, and for $C = A$ and $C = A^H$, we

- generate a consistent right-hand side B such that X is in the range space of C, compute a matrix X using PxGELS, and compute the ratio

$$||AX - B||/(\max(m,n)||A||||X||\epsilon)$$

- If C has more rows than columns (i.e. we are solving a least-squares problem), form $R = AX - B$, and check whether R is orthogonal to the column space of A by computing

$$||R^H C||/(\max(m,n,nrhs)||A||||B||\epsilon)$$

- If C has more columns than rows (i.e. we are solving an overdetermined system), check whether the solution X is in the row space of C by scaling both X and C to have norm one, and forming the QR factorization of $D = [A, X]$ if $C = A^H$, and the LQ factorization of $D = [A^H, X]^H$ if $C = A$. Letting $E = D(n : n+nrhs, n+1, n+nrhs)$ in the first case, and $E = D(m+1 : m+nrhs, m+1 : m+nrhs)$ in the latter, we compute

$$\max |d_{ij}| / (\max(m, n, nrhs)\epsilon)$$

3.6.1 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'
'MPI machine'
'LS.out'                          output file name (if any)
6                                 device out
3                                 number of problems sizes
55 17 31                          values of M
5 71 31                           values of N
3                                 number of NB's
2 3 5                             values of NB
3                                 number of NRHS's
2 3 5                             values of NRHS
2                                 number of NBRHS's
1 2                               values of NBRHS
4                                 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
4.0                               threshold
```

3.7 Tests for the ScaLAPACK INV routines

The inversion test driver tests five different matrix types – general nonsymmetric (GEN), general upper or lower triangular (UTR and LTR), and symmetric positive definite (upper or lower triangular) (UPD or LPD).

- If GEN, compute the LU factorization using PxGETRF, and then compute the inverse by invoking PxGETRI
- If UTR or LTR, set UPLO='U' or UPLO='L' respectively, and compute the inverse by invoking PxTRTRI
- If UPD or LPD, set UPLO='U' or UPLO='L' respectively, compute the Cholesky factorization using PxPOTRF, and then compute the inverse by invoking PxPOTRI
- Compute the ratio

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

3.7.1 Input File for Testing the ScaLAPACK INV Routines

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

```
'ScaLAPACK, Matrix Inversion Testing input file'
'MPI machine.'
'INV.out'                          output file name (if any)
6                                  device out
5                                  number of matrix types (next line)
'GEN' 'UTR' 'LTR' 'UPD' 'LPD'      GEN, UTR, LTR, UPD, LPD
4                                  number of problems sizes
2 5 10 15 13 20 30 50              values of N
4                                  number of NB's
2 3 4 5 6 20                       values of NB
4                                  number of process grids (ordered P & Q)
1 2 1 4 2 3 8                      values of P
1 1 4 1 3 2 1                      values of Q
1.0                                threshold
```

3.8 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

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

3.8.1 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'
'MPI 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
```

3.9 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

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

3.9.1 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'
'MPI 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
```

3.10 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

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

3.10.1 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'
'MPI 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
```

3.11 Tests for the ScaLAPACK SEP routines

The following tests will be performed on PxSYEV/PxHEEV, PxSYEVX/PxHEEVX and PxSYEVD/PxHEEVD:

$$r_1 = \frac{\|AZ - ZL\|}{abstol + ulp\ \|A\|}$$

$$r_2 = \frac{\|Z^* Z - I\|}{ulp\ \|A\|}$$

where Z is the matrix of eigenvectors returned when the eigenvector option is given, L is the matrix of eigenvalues, ulp represents PxLAMCH(ICTXT, 'P'), and $abstol$ represents $ulp * \|A\|$.

The 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 3.11.1.

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

3.11.1 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.

3.11.2 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 Test File'
' '
'sep.out'                       output file name (if any)
6                               device out (13 & 14 reserved for internal testing)
4  maximum number of processes
```

Type	Eigenvalue Distribution			
	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, ..., 2^i.

Table 1: Test matrices for the symmetric eigenvalue problem

```
'N'disable pxsyev tests, recommended for heterogeneous systems.
' '
'TEST 1 - test tiny matrices - different process configurations'
3                                number of matrices
0 1 2 matrix size
1 number of uplo choices
'L'uplo choices
2 number of processor configurations (P, Q, NB)
1 1           values of P (NPROW)
2 1   values of Q (NPCOL)
1 1   values of NB
1 number of matrix types
8    matrix types (see pdseptst.f)
'N'perform subset tests?
80.0 Threshold (* 5 for generalized tests)
-1  Absolute Tolerance
' '
'End of tests'
-1
```

3.12 Tests for the ScaLAPACK GSEP routines

Finding the eigenvalues and eigenvectors of symmetric matrices A and B, where B is also positive definite, follows the same stages as the symmetric eigenvalue problem except that the problem is first reduced from generalized to standard form using PxSYGST/PxHEGST.

To check these calculations, the following test ratios are computed:

$$\begin{aligned}
r_1 &= \frac{\|A\,Z - B\,Z\,D\|}{\|A\|\;\|Z\|\;n\,ulp} \\
&\quad \text{calling PxSYGVX/PxHEGVX with ITYPE=1 and UPLO='U'} \\
r_2 &= \frac{\|A\,Z - B\,Z\,D\|}{\|A\|\;\|Z\|\;n\,ulp} \\
&\quad \text{calling PxSYGVX/PxHEGVX with ITYPE=1 and UPLO='L'} \\
r_5 &= \frac{\|A\,Z - B\,Z\,D\|}{\|A\|\;\|Z\|\;n\,ulp} \\
&\quad \text{calling PxSYGVX/PxHEEVX with ITYPE=2 and UPLO='U'} \\
r_8 &= \frac{\|A\,B\,Z - Z\,D\|}{\|A\|\;\|Z\|\;n\,ulp} \\
&\quad \text{calling PxSYGVX/PxHEEVX with ITYPE=2 and UPLO='L'} \\
r_{10} &= \frac{\|A\,B\,Z - Z\,D\|}{\|A\|\;\|Z\|\;n\,ulp} \\
&\quad \text{calling PxSYGVX/PxHEEVX with ITYPE=3 and UPLO='U'} \\
r_{12} &= \frac{\|B\,A\,Z - Z\,D\|}{\|A\|\;\|Z\|\;n\,ulp} \\
&\quad \text{calling PxSYGVX/PxHEEVX with ITYPE=3 and UPLO='L'} \\
r_{14} &= \frac{\|B\,A\,Z - Z\,D\|}{\|A\|\;\|Z\|\;n\,ulp}
\end{aligned} \tag{1}$$

3.12.1 Input File for Testing the Generalized Symmetric Eigenvalue Routines and Drivers

The input file for testing the generalized symmetric eigenvalue routines and drivers is the same as that for testing the symmetric eigenproblem routines. Refer to the Section 3.11.2 for further details.

3.13 Tests for the ScaLAPACK NEP routines

The PxLAHQR test program generates random upper Hessenberg matrices, completes a Schur decomposition on them, and then tests the resulting Schur decomposition for maintaining similarity. The following tests will be performed on P_LAHQR:

$$\begin{aligned}
r_1 &= \frac{\left\|H - QSQ^T\right\|}{n\,ulp\;\|H\|} \\
r_2 &= \frac{\left\|I - Q^TQ\right\|}{n\,ulp}
\end{aligned} \tag{2}$$

where Q is the Schur vectors of the upper Hessenberg matrix H when the Schur vector and Schur decomposition option is given. N is the order of the matrix, ulp represents PxLAMCH(ICTXT, 'P'), and the one-norm is used for the norm computations.

3.13.1 Input File for Testing the ScaLAPACK NEP Routines

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

```
'SCALAPACK NEP (Nonsymmetric Eigenvalue Problem) input file'
'MPI Machine'
'NEP.out'                output file name (if any)
6                        device out
8                        number of problems sizes
1 2 3 4 6 10 100 200     values of N
3                        number of NB's
6 20 40                  values of NB
4                        number of process grids (ordered pairs of P & Q)
1 2 1 4                  values of P
1 2 4 1                  values of Q
20.0                     threshold
```

3.14 Tests for the ScaLAPACK EVC routines

The PCTREVC/PZTREVC test program performs a right and left eigenvector calculation of a triangular matrix followed by a residual checks of the calculated eigenvectors.

The following tests will be performed on P_TREVC. The basic test is:

$$r_1 = \frac{\|HZ - ZD\|}{n\,ulp\,\|T\|} \tag{3}$$

using the 1-norm. It also tests the normalization of Z.

$$r_2 = \frac{max_j\,\|m - norm(Z(j)) - 1\|)}{n\,ulp} \tag{4}$$

where H is the upper Hessenberg matrix, n is the order of the matrix, $Z(j)$ is the j-th eigenvector, and m-norm is the max-norm of a vector, and ulp represents PxLAMCH(ICTXT, 'P'). The max-norm of a complex n-vector x in this case is the maximum of $\|re(x(i))\| + \|im(x(i))\|$ over $i = 1, \ldots, n$.

3.14.1 Input File for Testing the ScaLAPACK EVC Routines

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

```
'SCALAPACK NEP (Nonsymmetric Eigenvalue Problem) input file'
'MPI Machine'
'EVC.out'                output file name (if any)
6                        device out
1                        number of problems sizes
100 1000 1500 2000 2500 3000    Probs
1                        number of NB's
8                       values of NB
4                        number of process grids (ordered pairs of P & Q)
1 1 4 2 3 2 2 1            values of P
1 4 1 2 3 1 4 8            values of Q
20.0                     threshold
```

3.15 Tests for the ScaLAPACK SVD routines

The following tests will be performed on PSGESVD/PDGESVD. A number of matrix "types" are specified, as denoted in Table 2. For each type of matrix, and for the minimal workspace as well as for larger than minimal workspace an M-by$N$ matrix "A" with known singular values is generated and used to test the SVD routines. For each matrix, A will be factored as $A = U\ diag(S)\ VT$ and the following 9 tests computed:

$$
\begin{aligned}
r_1 &= \frac{\|A - U1\mathrm{diag}(S1)VT1\|}{\|A\| \max(M,N)\, ulp} \\
r_2 &= \frac{\left\| I - (U1)^T U1 \right\|}{M\, ulp} \\
r_3 &= \frac{\left\| I - VT1(VT1)^T \right\|}{N\, ulp} \\
r_4 &= \begin{cases} 0 & \text{if } S1 \text{ contains SIZE nonnegative values in decreasing order.} \\ \frac{1}{ulp} & \text{otherwise} \end{cases} \\
r_5 &= \frac{\|S1 - S2\|}{SIZE\, M\, \|S\|} \\
r_6 &= \frac{\|U1 - U2\|}{M\, ulp} \\
r_7 &= \frac{\|S1 - S3\|}{SIZE\, ulp\, \|S\|} \\
r_8 &= \frac{\|VT1 - VT3\|}{N\, ulp} \\
r_9 &= \frac{\|S1 - S4\|}{SIZE\, ulp\, \|S\|}
\end{aligned}
$$

where ulp represents PxLAMCH(ICTXT, 'P').

3.15.1 Test Matrices for the Singular Value Decomposition Routines

Six different types of test matrices may be generated for the singular value decomposition routines. Table 2 shows the types available, along with the numbers used to refer to the matrix types. Except as noted, all matrix types other than the random bidiagonal matrices have $O(1)$ entries. The expression UDV means a real diagonal matrix D with $O(1)$ entries multiplied by unitary (or real orthogonal) matrices on the left and right.

	Singular Value Distribution	
Type	Arithmetic	Other
Zero		1
Identity		2
Diagonal	3	
UDV	4, $5^{\dagger}$, $6^{\ddagger}$	

†– matrix entries are $O(\sqrt{\text{overflow}})$
‡– matrix entries are $O(\sqrt{\text{underflow}})$

Table 2: Test matrices for the singular value decomposition

3.15.2 Input File for Testing the ScaLAPACK SVD Routines

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

```
'ScaLAPACK Singular Value Decomposition  input file'
6                               device out
4 maxnodes
' '
'TEST 1 - test medium matrices - all types and requests'
20.0                            Threshold
1                               number of matrices
100                             number of rows
25                              number of columns
1                               number of processor configurations (P, Q, NB)
2                               values of P (NPROW)
2                               values of Q (NPCOL)
8                               values of NB
' '
'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:

DB	general band (diagonally-dominant like)
DT	general tridiagonal (diagonally-dominant like)
GB	general band
GE	general (i.e. unsymmetric, in some cases rectangular)
GG	general matrices, generalized problem (i.e. a pair of general matrices)
HE	(complex) Hermitian
OR	(real) orthogonal
PB	symmetric or Hermitian positive definite band
PO	symmetric or Hermitian positive definite
PT	symmetric or Hermitian positive definite tridiagonal
ST	symmetric tridiagonal
SY	symmetric
TR	triangular (or in some cases quasi-triangular)
TZ	trapezoidal
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
EDC	compute eigenvectors using divide and conquer
EIN	compute selected eigenvectors by inverse iteration
EQU	equilibrate a matrix to reduce its condition number
EVC	compute the eigenvectors from the Schur factorization
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
GST	reduce a symmetric-definite generalized eigenvalue problem to standard form
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
MRZ	multiply by the orthogonal/unitary matrix from PxTZRZF
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
RZF	compute an RZ 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	DB	GB	DT	GT	PO	PB	PT	HE SY	TR	TZ	UN OR
TRF	×		×	×	×		×	×	×				
TRS	×		×	×	×		×	×	×		×		
RFS	×						×				×		
TRI	×						×				×		
CON	×						×				×		
EQU	×						×						
QPF	×												
QRF†	×	×											
RZF												×	
GQR†													×
MQR‡													×

†– also RQ, QL, and LQ

‡– also RQ, RZ, QL, and LQ

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

	GE	GG	HS	HG	TR	TG	HE SY	ST	PT	BD
HRD	×									
TRD							×			
BRD	×									
EQZ										
EIN								×		
EBZ								×		
EDC								×		
EVC					×			×		
GST							×			

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

	UN OR
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
EVD	compute all eigenvalues and, optionally, eigenvectors (using divide and conquer algorithm)
EVX	compute selected eigenvalues and eigenvectors
GVX	compute selected generalized eigenvalues and/or generalized eigenvectors
SVD	compute the SVD and/or singular vectors

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

	GE	GG	DB	GB	DT	GT	PO	PB	PT	HE SY	HB SB	ST
SV	$\times$		$\times$	$\times$	$\times$		$\times$	$\times$	$\times$			
SVX	$\times$						$\times$					
LS	$\times$											
EV										$\times$		
EVD										$\times$		
EVX										$\times$		
GVX										$\times$		
SVD	$\times$											

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_ORML2	(real) multiply by the orthogonal matrix from PxGELQF
P_ORMR2	(real) multiply by the orthogonal matrix from PxGERQF
P_ORMR3	(real) multiply by the orthogonal matrix from PxTZRZF

P_POTF2	compute the Cholesky factorization of a positive definite matrix
P_SYGS2	(real) reduce a symmetric-definite generalized eigenvalue problem to
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
P_UNMR3	(complex) multiply by the unitary matrix from PxTZRZF

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
P_LACHKIEEE	(real) 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_LACONSB	(real) looks for two consecutive small subdiagonal elements
P_LACP2	copies all or part of a distributed matrix to another distributed matrix
P_LACP3	(real) copies from a global parallel array into a local replicated array or vice versa.
P_LACPY	copy all or part of a distributed matrix to another distributed matrix
P_LAED0	Used by PxSTEDC.
P_LAED1	(real) Used by PxSTEDC.
P_LAED2	(real) Used by PxSTEDC.
P_LAED3	(real) Used by PxSTEDC.
P_LAEDZ	(real) Used by PxSTEDC.
P_LAEVSWP	moves the eigenvectors from where they are computed to a standard block cyclic array
P_LAHEF	(complex) compute part of the diagonal pivoting factorization of a Hermitian matrix
P_LAHQR	Find the Schur factorization of a Hessenberg matrix (modified version of HQR from EISPACK)
P_LAHRD	reduce NB columns of a general matrix to Hessenberg form
P_LAIECTB	(real) computes the number of negative eigenvalues in $(A - \Sigma I)$ where the sign bit is assumed to be bit 32.
P_LAIECTL	(real) computes the number of negative eigenvalues in $(A - \Sigma I)$ where the sign bit is assumed to be bit 64.
_LANV2	(complex) computes the Schur factorization of a real 2-by-2 nonsymmetric matrix
P_LAPIV	applies permutation matrix to a general distributed matrix
P_LAPV2	pivoting
P_LAQGE	equilibrate a general matrix

P_LAQSY	equilibrate a symmetric matrix
P_LARED1D	(real) Redistributes an array assuming that the input array, BYCOL, is distributed across rows and that all process columns contain the same copy of BYCOL.
P_LARED2D	Redistributes an array assuming that the input array, BYROW, is distributed across columns and that all process rows contain the same copy of BYROW. The output array, BYALL, will be identical on all processes.
P_LARF	apply (multiply by) an elementary reflector to a general rectangular matrix.
P_LARFB	apply (multiply by) a block reflector or its transpose/ conjugate-transpose to a general rectangular matrix.
P_LARFC	(complex) apply (multiply by) the conjugate-transpose of an elementary reflector to a general matrix.
P_LARFG	generate an elementary reflector (Householder matrix).
P_LARFT	form the triangular factor of a block reflector
P_LARZ	apply (multiply by) an elementary reflector as returned by P_TZRZF to a general matrix.
P_LARZB	apply (multiply by) a block reflector or its transpose/ conjugate transpose as returned by P_TZRZF to a general matrix.
P_LARZC	(complex) apply (multiply by) the conjugate transpose of an elementary reflector as returned by P_TZRZF to a general matrix.
P_LARZT	form the triangular factor of a block reflector as returned by P_TZRZF.
P_LASCL	multiplies a general rectangular matrix by a real scalar CTO/CFROM
P_LASE2	
P_LASET	initializes a matrix to BETA on the diagonal and ALPHA on the off-diagonals
P_LASMSUB	(real) looks for a small subdiagonal element from the bottom of the matrix that it can safely set to zero.
P_LASNBT	computes the position of the sign bit of a double precision floating point number
P_LASRT	
P_LASSQ	Compute a scaled sum of squares of the elements of a vector
P_LASWP	Perform a series of row interchanges
P_LATRA	computes the trace of a distributed matrix
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_LATRZ	reduces an upper trapezoidal matrix to upper triangular form
P_LAUU2	Unblocked version of P_LAUUM
P_LAUUM	Compute the product U*U' or L'*L (blocked version)
P_LAWIL	forms the Wilkinson transform

Bibliography

[1] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.

[2] J. Choi, J. Dongarra, S. Ostrouchov, A. Petitet, D. Walker, and R. C. Whaley, *A proposal for a set of parallel basic linear algebra subprograms*, Computer Science Dept. Technical Report CS-95-292, University of Tennessee, Knoxville, TN, May 1995. (Also LAPACK Working Note #100).

[3] ——, *The design and implementation of the ScaLAPACK LU, QR, and Cholesky factorization routines*, Scientific Programming, 5 (1996), pp. 173–184. (Also LAPACK Working Note #80).

[4] J. Dongarra and R. C. Whaley, *A user's guide to the BLACS v1.1*, Computer Science Dept. Technical Report CS-95-281, University of Tennessee, Knoxville, TN, 1995. (Also LAPACK Working Note #94).

[5] J. J. Dongarra, J. Du Croz, I. S. Duff, and S. Hammarling, *A set of Level 3 Basic Linear Algebra Subprograms*, ACM Trans. Math. Soft., 16 (1990), pp. 1–17.

[6] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson, *An extended set of FORTRAN basic linear algebra subroutines*, ACM Trans. Math. Soft., 14 (1988), pp. 1–17.

[7] M. P. I. Forum, *MPI: A message passing interface standard*, International Journal of Supercomputer Applications and High Performance Computing, 8 (1994), pp. 3–4. Special issue on MPI. Also available electronically, the URL is `ftp://www.netlib.org/mpi/mpi-report.ps` .

[8] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing*, MIT Press, Cambridge, MA, 1994.

[9] C. L. Lawson, R. J. Hanson, D. Kincaid, and F. T. Krogh, *Basic linear algebra subprograms for Fortran usage*, ACM Trans. Math. Soft., 5 (1979), pp. 308–323.

[10] R. C. Whaley, *Basic linear algebra communication subprograms: Analysis and implementation across multiple parallel architectures*, Computer Science Dept. Technical

Report CS-94-234, University of Tennessee, Knoxville, TN, May 1994. (Also LAPACK Working Note 73).