ScaLAPACK Tutorial *

Jack Dongarra[1,2] and L. Susan Blackford**[1]

[1] Department of Computer Science, University of Tennessee, Knoxville, TN 37996-1301, USA

[2] Mathematical Sciences Section, Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA

Abstract. ScaLAPACK is a library of high performance linear algebra routines for distributed memory MIMD computers. It is a continuation of the LAPACK project, which designed and produced analogous software for workstations, vector supercomputers, and shared memory parallel computers. The goals of the project are efficiency (to run as fast as possible), scalability (as the problem size and number of processors grow), reliability (including error bounds), portability (across all important parallel machines), flexibility (so users can construct new routines from well-designed parts), and ease-of-use (by making LAPACK and ScaLAPACK look as similar as possible). Many of these goals, particularly portability, are aided by developing and promoting *standards*, especially for low-level communication and computation routines. We have been successful in attaining these goals, limiting most machine dependencies to two standard libraries called the BLAS, or Basic Linear Algebra Subroutines, and BLACS, or Basic Linear Algebra Communication Subroutines. ScaLAPACK will run on any machine where both the BLAS and the BLACS are available.

This tutorial will begin by reviewing the fundamental design principles of the BLAS and LAPACK and their influence on the development of ScaLAPACK. The two dimensional block cyclic data decomposition will be presented, followed by a discussion of the underlying building blocks of ScaLAPACK, the BLACS and the PBLAS. The contents of the ScaLAPACK library will then be enumerated, followed by example programs and performance results. And finally, future directions and related projects will be described.

1 Introduction

Much of the work in developing linear algebra software for advanced architecture computers is motivated by the need to solve large problems on the fastest

---

** formerly L. Susan Ostrouchov

* 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.

computers available. In this tutorial, we focus on the development of standards for use in linear algebra and the building blocks for a library and the aspects of algorithmic design and parallel implementation.

The linear algebra community has long recognized the need for help in developing algorithms into software libraries, and several years ago, as a community effort, put together a *de facto* standard for identifying basic operations required in linear algebra algorithms and software. The hope was that the routines making up this standard, the Basic Linear Algebra Subprograms (BLAS), would be implemented on advanced-architecture computers by many manufacturers, making it possible to reap the portability benefits of having them efficiently implemented on a wide range of machines. This goal has been largely realized.

The key insight of our approach to designing linear algebra algorithms for advanced architecture computers is that the frequency with which data are moved between different levels of the memory hierarchy must be minimized in order to attain high performance. Thus, our main algorithmic approach for exploiting both vectorization and parallelism is the use of block-partitioned algorithms, particularly in conjunction with highly-tuned kernels for performing matrix-vector and matrix-matrix operations (BLAS). In general, block-partitioned algorithms require the movement of blocks, rather than vectors or scalars, resulting in a greatly reduced startup cost because fewer messages are exchanged.

A second key idea is that the performance of an algorithm can be tuned by a user by varying the parameters that specify the data layout. On shared memory machines, this is controlled by the block size, while on distributed memory machines it is controlled by the block size and the configuration of the logical process grid.

Sections 2 and 3 review the fundamental design principles of the BLAS and LAPACK. The two dimensional block cyclic data decomposition is presented in Section 4 as the basis for matrix distribution in ScaLAPACK. The building blocks of the ScaLAPACK library, the BLACS and the PBLAS, are then presented in Sections 5.1 and 5.2. The contents and performance of ScaLAPACK and described in Section 5. And finally, Section 6 summarizes and discusses future directions and related projects.

2 The Basic Linear Algebra Subprograms (BLAS)

The Basic Linear Algebra Subprograms are key to portability and efficiency across sequential and parallel environments. There are three levels of BLAS:

Level 1 BLAS [2]: for vector operations, such as $y \leftarrow \alpha x + y$
Level 2 BLAS [3]: for matrix-vector operations, such as $y \leftarrow \alpha Ax + \beta y$
Level 3 BLAS [4]: for matrix-matrix operations, such as $C \leftarrow \alpha AB + \beta C$.

Here, A, B and C are matrices, x and y are vectors, and α and β are scalars.

The Level 1 BLAS are used in LAPACK, but for convenience rather than for performance: they perform an insignificant fraction of the computation, and they cannot achieve high efficiency on most modern supercomputers.

The Level 2 BLAS can achieve near-peak performance on many vector processors, such as a CRAY Y-MP, or Convex C-2 machine. However, on other vector processors such as a CRAY-2, the performance of the Level 2 BLAS is limited by the rate of data movement between different levels of memory [11].

The Level 3 BLAS overcome this limitation. They perform $O(n^3)$ floating-point operations on $O(n^2)$ data, whereas the Level 2 BLAS perform only $O(n^2)$ operations on $O(n^2)$ data. The Level 3 BLAS also allow us to exploit parallelism in a way that is transparent to the software that calls them. While the Level 2 BLAS offer some scope for exploiting parallelism, greater scope is provided by the Level 3 BLAS, as Table 1 illustrates.

Table 1. Speed (Megaflops) of BLAS Operations on a CRAY Y-MP. All matrices are of order 500.

Number of processors:	1	2	4	8
Level 2: $y \leftarrow \alpha Ax + \beta y$	311	611	1197	2285
Level 3: $C \leftarrow \alpha AB + \beta C$	312	623	1247	2425
Peak	333	666	1332	2664

3 The Linear Algebra Package (LAPACK)

LAPACK [1] provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems. The associated matrix factorizations (LU, Cholesky, QR, SVD, Schur, generalized Schur) are also provided, as are related computations such as reordering of the Schur factorizations and estimating condition numbers. Dense and banded matrices are handled, but not general sparse matrices. In all areas, similar functionality is provided for real and complex matrices, in both single and double precision.

The original goal of the LAPACK project was to make the widely used EISPACK [17] and LINPACK libraries run efficiently on shared-memory vector and parallel processors. On these machines, LINPACK and EISPACK are inefficient because their memory access patterns disregard the multi-layered memory hierarchies of the machines, thereby spending too much time moving data instead of doing useful floating-point operations. LAPACK addresses this problem by reorganizing the algorithms to use block matrix operations, such as matrix multiplication, in the innermost loops [6, 1]. These block operations can be optimized for each architecture to account for the memory hierarchy [5], and so provide a transportable way to achieve high efficiency on diverse modern machines.

LAPACK can be regarded as a successor to LINPACK and EISPACK. It has virtually all the capabilities of these two packages and much more besides. It improves on them in four main respects: speed, accuracy, robustness and functionality. While LINPACK and EISPACK are based on the vector operation kernels of the Level 1 BLAS, LAPACK was designed at the outset to exploit the matrix-matrix operation kernels of the Level 3 BLAS. Because of the coarse granularity of these operations, their use tends to promote high efficiency on many high-performance computers, particularly if specially coded implementations are provided by the manufacturer.

Extensive performance results for LAPACK can be found in the LAPACK Users' Guide [1].

3.1 A Block Partitioned Algorithm Example

We consider the Cholesky factorization algorithm, which factorizes a symmetric positive definite matrix as $A = U^T U$. To derive a block form of Cholesky factorization, we partition the matrices into blocks, in which the diagonal blocks of A and U are square, but of differing sizes. We assume that the first block has already been factored as $A_{00} = U_{00}^T U_{00}$, and that we now want to determine the second block column of U consisting of the blocks U_{01} and U_{11}.

$$\begin{pmatrix} A_{00} & A_{01} & A_{02} \\ . & A_{11} & A_{12} \\ . & . & A_{22} \end{pmatrix} = \begin{pmatrix} U_{00}^T & 0 & 0 \\ U_{01}^T & U_{11}^T & 0 \\ U_{02}^T & U_{12}^T & U_{22}^T \end{pmatrix} \begin{pmatrix} U_{00} & U_{01} & U_{02} \\ 0 & U_{11} & U_{12} \\ 0 & 0 & U_{22} \end{pmatrix}.$$

Equating submatrices in the second block of columns, we obtain

$$\begin{aligned} A_{01} &= U_{00}^T U_{01} \\ A_{11} &= U_{01}^T U_{01} + U_{11}^T U_{11}. \end{aligned}$$

Hence, since U_{00} has already been computed, we can compute U_{01} as the solution to the equation

$$U_{00}^T U_{01} = A_{01}$$

by a call to the Level 3 BLAS routine STRSM; and then we can compute U_{11} from

$$U_{11}^T U_{11} = A_{11} - U_{01}^T U_{01}.$$

This involves first updating the symmetric submatrix A_{11} by a call to the Level 3 BLAS routine SSYRK, and then computing its Cholesky factorization. Since Fortran 77 does not allow recursion, a separate routine must be called (using Level 2 BLAS rather than Level 3), named SPOTF2 in Figure 1. In this way, successive blocks of columns of U are computed. The LAPACK-style code for the block algorithm is shown in Figure 1.

```
do j = 0, n-1, nb
  jb = min( nb, n-j )
  call strsm( 'left', 'upper', 'transpose', 'non-unit', j, jb, one,
              a, lda, a(0,j), lda )
  call ssyrk( 'upper', 'transpose', jb, j, -one, a(0,j), lda, one,
              a(j,j), lda )
  call spotf2( 'upper', jb, a(j,j), lda, info )
  if( info .ne. 0 ) go to 20
end do
```

Fig. 1. The body of the "LAPACK-style" routine for block Cholesky factorization. In this code fragment, nb denotes the width of the blocks.

4 Block Cyclic Data Distribution

The way in which a matrix is distributed over the processes has a major impact on the load balance and communication characteristics of the concurrent algorithm, and hence largely determines its performance and scalability. The block cyclic distribution provides a simple, yet general-purpose way of distributing a block-partitioned matrix on distributed memory concurrent computers. It has been incorporated in the High Performance Fortran standard [14].

The block cyclic data distribution is parameterized by the four numbers P_r, P_c, r, and c, where $P_r \times P_c$ is the process template and $r \times c$ is the block size.

Suppose first that we have M objects, indexed by an integer $0 \le m < M$, to map onto P processes, using block size r. The m-th item will be stored in the i-th location of block b on process p, where

$$\langle p, b, i \rangle = \left\langle \left\lfloor \frac{m}{r} \right\rfloor \bmod P,\ \left\lfloor \frac{\lfloor \frac{m}{r} \rfloor}{P} \right\rfloor,\ m \bmod r \right\rangle \ .$$

In the special case where $r = 2^{\hat{r}}$ and $P = 2^{\hat{P}}$ are powers of two, this mapping is really just bit extraction, with i equal to the rightmost $\hat{r}$ bits of m, p equal to the next $\hat{P}$ bits of m, and b equal to the remaining leftmost bits of m. The distribution of a block-partitioned matrix can be regarded as the tensor product of two such mappings: one that distributes the rows of the matrix over P_r processes, and another that distributes the columns over P_c processes. That is, the matrix element indexed globally by (m, n) is stored in location

$$\langle (p,q), (b,d), (i,j) \rangle =$$

$$\left\langle \left(\left\lfloor \frac{m}{r} \right\rfloor \bmod P_r,\ \left\lfloor \frac{n}{c} \right\rfloor \bmod P_c \right),\ \left(\left\lfloor \frac{\lfloor \frac{m}{r} \rfloor}{P_r} \right\rfloor,\ \left\lfloor \frac{\lfloor \frac{n}{c} \rfloor}{P_c} \right\rfloor \right),\ (m \bmod r,\ n \bmod c) \right\rangle \ .$$

The nonscattered decomposition (or pure block distribution) is just the special case $r = \lceil M/P_r \rceil$ and $c = \lceil N/P_c \rceil$. Similarly a purely scattered decomposition (or two dimensional wrapped distribution) is the special case $r = c = 1$.

5 ScaLAPACK

The ScaLAPACK software library is extending the LAPACK library to run scalably on MIMD, distributed memory, concurrent computers [9]. For such machines the memory hierarchy includes the off-processor memory of other processors, in addition to the hierarchy of registers, cache, and local memory on each processor. Like LAPACK, the ScaLAPACK routines are based on block-partitioned algorithms in order to minimize the frequency of data movement between different levels of the memory hierarchy. The fundamental building blocks of the ScaLAPACK library are a set of Basic Linear Algebra Communication Subprograms (BLACS) [8] for communication tasks that arise frequently in parallel linear algebra computations, and the Parallel Basic Linear Algebra Subprograms (PBLAS), which are a distributed memory version of the sequential BLAS. In the ScaLAPACK routines, all interprocessor communication occurs within the PBLAS and the BLACS, so the source code of the top software layer of ScaLAPACK looks very similar to that of LAPACK.

Figure 2 describes the ScaLAPACK software hierarchy. The components below the dashed line, labeled Local, are called on a single processor, with arguments stored on single processors only. The components above the line, labeled Global, are synchronous parallel routines, whose arguments include matrices and vectors distributed in a 2D block cyclic layout across multiple processors.

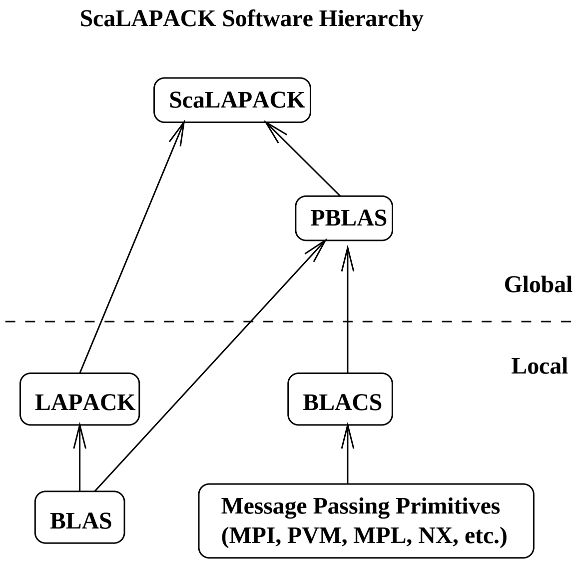

Fig. 2. ScaLAPACK Software Hierarchy

5.1 The Basic Linear Algebra Communication Subprograms (BLACS)

The **BLACS** (Basic Linear Algebra Communication Subprograms) [8] are a message passing library designed for linear algebra. The computational model consists of a one or two dimensional grid of processes, where each process stores matrices and vectors. The BLACS include synchronous send/receive routines to send a matrix or submatrix from one process to another, to broadcast submatrices to many processes, or to compute global reductions (sums, maxima and minima). There are also routines to construct, change, or query the process grid. Since several ScaLAPACK algorithms require broadcasts or reductions among different subsets of processes, the BLACS permit a process to be a member of several overlapping or disjoint process grids, each one labeled by a *context*. Some message passing systems, such as MPI [15], also include this context concept. The BLACS provide facilities for safe inter-operation of system contexts and BLACS contexts.

5.2 PBLAS

In order to simplify the design of ScaLAPACK, and because the BLAS have proven to be very useful tools outside LAPACK, we chose to build a Parallel BLAS, or PBLAS [10], whose interface is as similar to the BLAS as possible. This decision has permitted the ScaLAPACK code to be quite similar, and sometimes nearly identical, to the analogous LAPACK code. Only one substantially new routine was added to the PBLAS, matrix transposition, since this is a complicated operation in a distributed memory environment [7].

We hope that the PBLAS will provide a distributed memory standard, just as the BLAS have provided a shared memory standard. This would simplify and encourage the development of high performance and portable parallel numerical software, as well as providing manufacturers with a small set of routines to be optimized. The acceptance of the PBLAS requires reasonable compromises among competing goals of functionality and simplicity.

The PBLAS operate on matrices distributed in a 2D block cyclic layout. Since such a data layout requires many parameters to fully describe the distributed matrix, we have chosen a more object-oriented approach, and encapsulated these parameters in an integer array called an *array descriptor*. An array descriptor includes

(1) the descriptor type,
(2) the BLACS context (see Section 5.1),
(3) the number of rows in the distributed matrix,
(4) the number of columns in the distributed matrix,
(5) the row block size (r in Section 4),
(6) the column block size (c in Section 4),
(7) the process row over which the first row of the matrix is distributed,
(8) the process column over which the first column of the matrix is

distributed,
(9) the leading dimension of the local array storing the local blocks.

By using this descriptor, a call to a PBLAS routine is very similar to a call to the corresponding BLAS routine.

```
CALL DGEMM ( TRANSA, TRANSB, M, N, K, ALPHA,
             A( IA, JA ), LDA,
             B( IB, JB ), LDB, BETA,
             C( IC, JC ), LDC )

CALL PDGEMM( TRANSA, TRANSB, M, N, K, ALPHA,
             A, IA, JA, DESC_A,
             B, JB, DESC_B, BETA,
             C, IC, JC, DESC_C )
```

`DGEMM` computes `C = BETA * C + ALPHA * op( A ) * op( B )`, where `op(A)` is either `A` or its transpose depending on `TRANSA`, `op(B)` is similar, `op(A)` is `M`-by-`K`, and `op(B)` is `K`-by-`N`. `PDGEMM` is the same, with the exception of the way in which submatrices are specified. To pass the submatrix starting at `A(IA,JA)` to `DGEMM`, for example, the actual argument corresponding to the formal argument `A` would simply be `A(IA,JA)`. `PDGEMM`, on the other hand, needs to understand the global storage scheme of `A` to extract the correct submatrix, so `IA` and `JA` must be passed in separately. `DESC_A` is the array descriptor for `A`. The parameters describing the matrix operands `B` and `C` are analogous to those describing `A`. In a truly object-oriented environment matrices and `DESC_A` would be synonymous. However, this would require language support, and detract from portability.

The presence of a context associated with every distributed matrix provides the ability to have separate "universes" of message passing. The use of separate communication contexts by distinct libraries (or distinct library invocations) such as the PBLAS insulates communication internal to the library from external communication. When more than one descriptor array is present in the argument list of a routine in the PBLAS, it is required that the individual BLACS context entries must be equal. In other words, the PBLAS do not perform "inter-context" operations.

We have not included specialized routines to take advantage of packed storage schemes for symmetric, Hermitian, or triangular matrices, nor of compact storage schemes for banded matrices [10].

5.3 ScaLAPACK sample code

Given the infrastructure described above, the ScaLAPACK version (PDGETRF) of the LU decomposition is nearly identical to its LAPACK version (DGETRF).

SEQUENTIAL LU FACTORIZATION CODE

```
      DO 20 J = 1, MIN( M, N ), NB
         JB = MIN( MIN( M, N )-J+1, NB )

         Factor diagonal and subdiagonal blocks and test for exact
         singularity.

         CALL DGETF2( M-J+1, JB, A( J, J ), LDA, IPIV( J ),
     $                IINFO )

         Adjust INFO and the pivot indices.

         IF( INFO.EQ.0 .AND. IINFO.GT.0 ) INFO = IINFO + J - 1
         DO 10 I = J, MIN( M, J+JB-1 )
            IPIV( I ) = J - 1 + IPIV( I )
   10    CONTINUE

         Apply interchanges to columns 1:J-1.

         CALL DLASWP( J-1, A, LDA, J, J+JB-1, IPIV, 1 )


         IF( J+JB.LE.N ) THEN

            Apply interchanges to columns J+JB:N.

            CALL DLASWP( N-J-JB+1, A( 1, J+JB ), LDA, J, J+JB-1,
     $                   IPIV, 1 )

            Compute block row of U.

            CALL DTRSM( 'Left', 'Lower', 'No transpose', 'Unit',
     $                  JB, N-J-JB+1, ONE, A( J, J ), LDA,
     $                  A( J, J+JB ), LDA )
            IF( J+JB.LE.M ) THEN

               Update trailing submatrix.

               CALL DGEMM( 'No transpose', 'No transpose',
     $                     M-J-JB+1, N-J-JB+1, JB, -ONE,
     $                     A( J+JB, J ), LDA, A( J, J+JB ), LDA,
     $                     ONE, A( J+JB, J+JB ), LDA )
            END IF
         END IF
   20 CONTINUE
```

PARALLEL LU FACTORIZATION CODE

```
      DO 10 J = JA, JA+MIN(M,N)-1, DESCA( 5 )
         JB = MIN( MIN(M,N)-J+JA, DESCA( 5 ) )
         I = IA + J - JA

         Factor diagonal and subdiagonal blocks and test for exact
         singularity.

         CALL PDGETF2( M-J+JA, JB, A, I, J, DESCA, IPIV, IINFO )


         Adjust INFO and the pivot indices.

         IF( INFO.EQ.0 .AND. IINFO.GT.0 )
     $      INFO = IINFO + J - JA



         Apply interchanges to columns JA:J-JA.

         CALL PDLASWP( 'Forward', 'Rows', J-JA, A, IA, JA, DESCA,
     $                 J, J+JB-1, IPIV )

         IF( J-JA+JB+1.LE.N ) THEN

            Apply interchanges to columns J+JB:JA+N-1.

            CALL PDLASWP( 'Forward', 'Rows', N-J-JB+JA, A, IA,
     $                    J+JB, DESCA, J, J+JB-1, IPIV )

            Compute block row of U.

            CALL PDTRSM( 'Left', 'Lower', 'No transpose', 'Unit',
     $                   JB, N-J-JB+JA, ONE, A, I, J, DESCA, A, I,
     $                   J+JB, DESCA )
            IF( J-JA+JB+1.LE.M ) THEN

               Update trailing submatrix.

               CALL PDGEMM( 'No transpose', 'No transpose',
     $                      M-J-JB+JA, N-J-JB+JA, JB, -ONE, A,
     $                      I+JB, J, DESCA, A, I, J+JB, DESCA,
     $                      ONE, A, I+JB, J+JB, DESCA )
            END IF
         END IF
   10 CONTINUE
```

5.4 ScaLAPACK – Contents and Documentation

The ScaLAPACK library provides routines for the solution of linear systems of equations, symmetric positive definite banded linear systems of equations, condition estimation and iterative refinement, for LU and Cholesky factorization, matrix inversion, full-rank linear least squares problems, orthogonal and generalized orthogonal factorizations, orthogonal transformation routines, reductions to upper Hessenberg, bidiagonal and tridiagonal form, reduction of a symmetric-definite generalized eigenproblem to standard form, the symmetric, generalized symmetric and the nonsymmetric eigenproblem. Similar functionality is provided for real and complex matrices, in both single and double precision.

A comprehensive Installation Guide is provided, as well as test suites for all ScaLAPACK, PBLAS, and BLACS routines.

5.5 ScaLAPACK – Performance

The main factors that affect the performance of linear algebra software on distributed-memory machines are the block size and the configuration of the process grid.

The ScaLAPACK codes run efficiently on a wide range of distributed memory MIMD computers, such as the IBM SP series, the Cray T3 series, and the Intel series. Figure 3 presents a variety of performance results on the IBM SP-2 and the Intel Paragon. Extensive performance results can be found in [9]. On 8 wide nodes of an IBM SP-2 for example, a 13000×13000 LU factorization runs at 1.6 Gflop/s. A 2000×2000 LU factorization on the same machine reaches already 1.0 Gflop/s. These performance results correspond to a very efficient use of this machine. The ScaLAPACK library can also be used on clusters of workstations, and any system for which PVM [12] or MPI [13, 15, 16] is available. Performance results on the TMC CM-5, however, have been disappointing because of the difficulty of using the vector units in message passing programs.

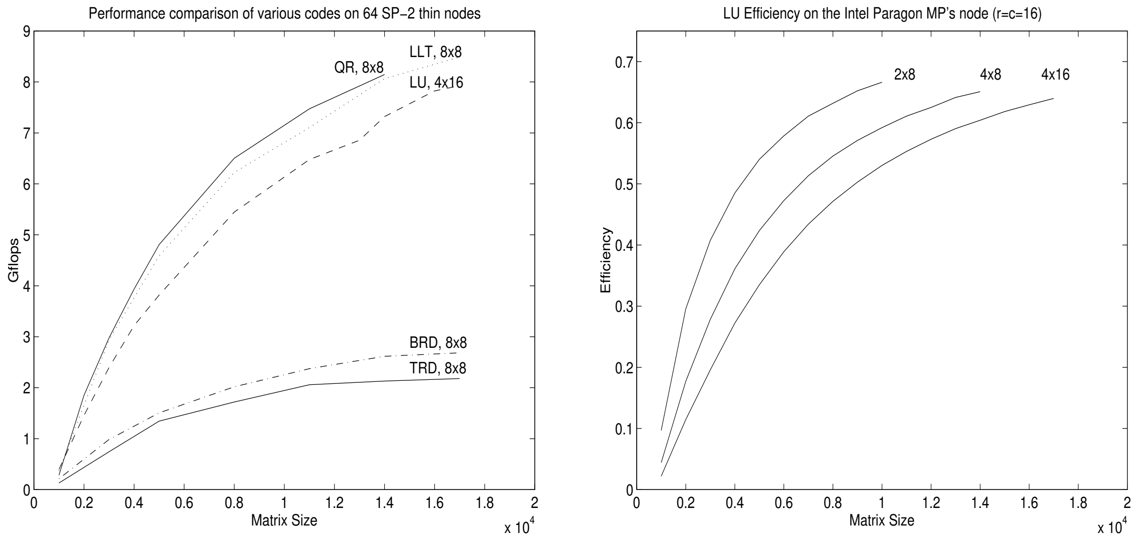

Fig. 3. IBM SP-2 and Intel Paragon Performance

6 Conclusions and Related Projects

Both LAPACK and ScaLAPACK are slated for updated software and documentation releases toward the end of 1996. Alternative interfaces for the libraries are under development. A BLAS Technical Forum is being established to consider expanding the BLAS for serial and parallel computation. And finally, a number of other aspects of the ScaLAPACK project, including sparse and out-of-core activities, are available or underway.

The upcoming LAPACK release (version 3.0) will introduce routines for the singular value decomposition computed by the divide-and-conquer method, new simple and expert drivers for the generalized nonsymmetric eigenproblem, a faster QR decomposition with column pivoting, a faster solver for the rank-deficient least squares, and a blocked version of the reduction of a upper trapezoidal matrix to upper triangular form. The third edition of the LAPACK Users' Guide will coincide with this release.

Future releases of the ScaLAPACK library will extend the flexibility of the PBLAS and increase the functionality of the library to include routines for the solution of general banded linear systems, general and symmetric positive definite tridiagonal systems, rank-deficient linear least squares problems, generalized linear least squares problems, and the singular value decomposition. A draft of the ScaLAPACK Users' Guide is currently available on *netlib* and we plan to have the final draft ready for publication at the end of 1996.

A Fortran 90 interface for the LAPACK library is currently under development. This interface will provide an improved user-interface to the package by taking advantage of the considerable simplifications of the Fortran 90 language such as assumed-shape arrays, optional arguments, and generic interfaces.

As HPF compilers have recently become available, work is currently in progress to produce an HPF interface for the ScaLAPACK library. HPF provides a much more convenient distributed memory interface than can be provided in processor-level languages such as Fortran77 or C. This interface to a subset of the ScaLAPACK routines will be available at the time of the next ScaLAPACK release. With the increased generality of the PBLAS to operate on partial first blocks, ScaLAPACK will be fully compatible with HPF [14].

Also underway is an effort to establish a BLAS Technical Forum to consider expanding the BLAS in a number of ways in light of modern software, language, and hardware developments. The goals of the forum are to stimulate thought and discussion, and define functionality and future development of a set of standards for basic matrix and data structures. Dense and sparse BLAS are considered, as well as calling sequences for a set of low-level computational kernels for the parallel and sequential settings. For more information on the BLAS Technical Forum refer to the URL:

http://www.netlib.org/utk/papers/blast-forum.html

The EISPACK, LINPACK, LAPACK, BLACS and SCALAPACK linear algebra libraries are in the public domain. The software and documentation can be retrieved from *netlib* (`http://www.netlib.org`).

7 Acknowledgments

We wish to acknowledge Antoine Petitet for assembling the previous version of this tutorial, and thank him for his advice and contributions to this effort.

References

1. Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S., Sorensen, D.: LAPACK

Users' Guide, Second Edition. SIAM, Philadelphia, PA, 1995.
2. Lawson, C., Hanson, R., Kincaid, D., Krogh, F.: Basic Linear Algebra Subprograms for Fortran Usage. ACM Transactions on Mathematical Software, 5:308–323, 1979.
3. Dongarra, J., Du Croz, J., Hammarling, S., Hanson, R.: Algorithm 656: An extended Set of Basic Linear Algebra Subprograms: Model Implementation and Test Programs. ACM Transactions on Mathematical Software, 14(1):18–32, 1988.
4. Dongarra, J., Du Croz, J., Duff, I., Hammarling, S.: A Set of Level 3 Basic Linear Algebra Subprograms. ACM Transactions on Mathematical Software, 16(1):1–17, 1990.
5. Anderson, E., Dongarra, J.: Results from the initial release of LAPACK. LAPACK working note 16, Computer Science Department, University of Tennessee, Knoxville, TN, 1989.
6. Anderson, E., Dongarra, J.: Evaluating block algorithm variants in LAPACK. LAPACK working note 19, Computer Science Department, University of Tennessee, Knoxville, TN, 1990.
7. Choi, J., Dongarra, J., Walker, D.: Parallel matrix transpose algorithms on distributed memory concurrent computers. In Proceedings of Fourth Symposium on the Frontiers of Massively Parallel Computation (McLean, Virginia), pages 245–252. IEEE Computer Society Press, Los Alamitos, California, 1993. (also LAPACK Working Note #65).
8. Dongarra, J., Whaley, R.C.: A User's Guide to the BLACS v1.0. Technical Report UT CS-95-281, LAPACK Working Note #94, University of Tennessee, 1995.
9. Choi, J., Demmel, J., Dhillon, I., Dongarra, J., Ostrouchov, S., Petitet, A., Stanley, K., Walker, D., Whaley, R.C.: ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers - Design Issues and Performance. Technical Report UT CS-95-283, LAPACK Working Note #95, University of Tennessee, 1995.
10. Choi, J., Dongarra, J., Ostrouchov, S., Petitet, A., Walker, D., Whaley, R.C.: A Proposal for a Set of Parallel Basic Linear Algebra Subprograms. Technical Report UT CS-95-292, LAPACK Working Note #100, University of Tennessee, 1995.
11. Dongarra, J., Duff, I., Sorensen, D., Van der Vorst, H.: Solving Linear Systems on Vector and Shared Memory Computers. SIAM Publications, Philadelphia, PA, 1991.
12. Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., V. Sunderam, V.: PVM: Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing. The MIT Press, Cambridge, Massachusetts, 1994.
13. Gropp, W., Lusk, E. Skjellum, A.: Using MPI: Portable Programming with the Message-Passing Interface, MIT Press, Cambridge, MA, 1994.
14. Koebel, C., Loveman, D., Schreiber, R., Steele, G., Zosel, M.: The High Performance Fortran Handbook. The MIT Press, Cambridge, Massachusetts, 1994.
15. Message Passing Interface Forum. MPI: A Message Passing Interface Standard. International Journal of Supercomputer Applications and High Performance Computing, 8(3–4), 1994.
16. Snir, M., Otto, S. W., Huss-Lederman, S., Walker, D. W. and Dongarra, J.: MPI: The Complete Reference, MIT Press, Cambridge, MA, 1996.
17. Wilkinson, J., Reinsch, C.: Handbook for Automatic Computation: Volume II - Linear Algebra. Springer-Verlag, New York, 1971.

This article was processed using the LaTeX macro package with LLNCS style