ScaLAPACK: A Linear Algebra Library for Message-Passing Computers *

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†

January 6, 1997

Abstract

This article outlines the content and performance of some of the ScaLAPACK software. ScaLAPACK is a collection of mathematical software for linear algebra computations on distributed-memory computers. The importance of developing standards for computational and message-passing interfaces is discussed. We present the different components and building blocks of ScaLAPACK and provide initial performance results for selected PBLAS routines and a subset of ScaLAPACK driver routines.

1 Introduction

ScaLAPACK is a library of high-performance linear algebra routines for distributed-memory MIMD machines. It is a continuation of the LAPACK project, which has designed and produced an efficient linear algebra library for workstations, vector supercomputers, and shared-memory parallel computers [3]. Both libraries contain routines for the solution of systems of linear equations, linear least squares problems, and eigenvalue problems. The goals of the LAPACK project, which continue into the ScaLAPACK project, are *efficiency,* so that the computationally intensive routines execute as fast as possible; *scalability* as the problem size and number of processors grow; *reliability,* including the return of error bounds; portability across machines; *flexibility* so that users may construct new routines from well-designed components; and *ease of use.* Toward this last goal the ScaLAPACK software has been designed to look as much like the LAPACK software as possible.

Many of these goals have been attained by developing and promoting standards, especially specifications for basic computational and communication routines. Thus,

*This work was supported in part by the Defense Advanced Research Projects Agency under contract DAAL03-91-C-0047, administered by the Army Research Office and by the Office of Scientific Computing, U.S. Department of Energy, under Contract DE-AC05-84OR21400.

†Department of Computer Science, University of Tennessee, Knoxville, TN 37996-1301

‡Soongsil University, Korea

§Mathematical Sciences Section, Oak Ridge National Laboratory, Oak Ridge, TN 37831

¶Computer Science Division, University of California - Berkeley, Berkeley, CA 94720

‖Department of Computer Science, University of Tennessee, Knoxville, TN 37996-1301, and Mathematical Sciences Section, Oak Ridge National Laboratory, Oak Ridge, TN 37831

**NAG Ltd, England

††Intel SSPD, 15201 NW Greenbrier Pkwy., Bldg. CO1-01, Beaverton, OR 97006-5733

‡‡University of Wales, Cardiff

LAPACK relies on the Basic Linear Algebra Subroutines BLAS [30, 16, 15], particularly the Level 2 and 3 BLAS, for computational efficiency; and ScaLAPACK [5] relies upon the Basic Linear Algebra Communications Subprograms (BLACS) [20] for efficiency of communication and uses a set of parallel BLAS, the PBLAS [9], which themselves call the BLAS and the BLACS. LAPACK and ScaLAPACK will run on any machine for which the BLAS and the BLACS are available. A PVM [23] version of the BLACS has been available for some time, and the portability of the BLACS has recently been further increased by the development of a version that uses MPI [33].

2 Factors Affecting ScaLAPACK Performance

Unlike software such as LAPACK, which is designed for sequential computers and shared-memory systems, the performance of the ScaLAPACK library is sensitive not only to the characteristics of the distributed-memory computer but also to the parameters of the data distribution. Choosing these distribution parameters is often perceived by the user as a difficult task. These degrees of freedom are, however, the fundamental factor that allows the ScaLAPACK software to be inherently and potentially highly efficient for a wide range of distributed-memory concurrent computers. In this article, we discuss the major factors that affect the performance of the ScaLAPACK driver routines. Also provided are guidelines to help users in their quest for high performance.

2.1 Processor Performance and Total Performance

The ScaLAPACK routines have been specifically designed to allow for an even distribution of the computational load and thus to achieve the highest possible performance. Therefore, the overall execution time is strongly related to the rate of floating-point operations per second (flop/s) that the slowest processor in the machine configuration can achieve.

This behavior can easily be observed if some factor slows a particular processor of the system. Consider, for instance, a ten-processor machine configuration. Suppose that nine of the processors can deliver a peak performance of 100 megaflop/s (Mflop/s) but that the tenth processor can achieve only 20 Mflop/s. (On a homogeneous system, different versions of the operating system and/or memory capacities, I/O traffic, or simply another user's program can easily cause such a performance degradation.) On such a ten-processor machine, the overall ScaLAPACK peak performance is thus limited to 200 Mflop/s, whereas the performance of the machine with nine 100-megaflop/s processors is 900 Mflop/s. Specifically, *the most heavily loaded processor controls execution time.* The implications are clear. If a user's code is running on nine unloaded processors and one processor with a load factor of 5, one can observe no more than a factor of $\frac{10}{5}$ speedup.

Similarly, it is possible on some systems to spawn multiple processes on a single processor. In such a case, performance is limited by the slowest processor, presumably the one with the most processes. For example, if 10 processes are spawned on 9 identical processors, the speedup is limited to $10/\lceil\frac{10}{9}\rceil$.

The load of the machine, in addition to the direct effect of offering a program only a portion of the total cycles, can have several indirect effects. If each processor is individually scheduled, performance can be arbitrarily poor because significant progress is possible only when all processes are concurrently scheduled. A loaded machine may also cause one's data to be swapped out to disk, which can greatly reduce peak performance.

2.2 ScaLAPACK and the BLAS

The ScaLAPACK software relies as much as possible on the BLAS for efficiency and portability. Consequently, the local processor flop rate can be best approximated (from the user's viewpoint) by the performance of the BLAS. The user is therefore strongly urged to use, whenever possible, the most efficient BLAS implementation available. Not using a machine-optimized BLAS implementation may substantially lower the peak flop rate that the hardware can achieve.

2.3 Data Distribution

The ScaLAPACK software assumes that the user's input data has been distributed on a two-dimensional grid of processes according to the block cyclic scheme. For a given number of processes, the parameters of this family of data distributions are the shape of the process grid and the size of the block used to partition and distribute the matrix entries over the process grid. These parameters affect the number of messages exchanged during the operation, the aggregated volume of data communicated, and the computational load balance. These factors have a significant impact on the efficiency achieved by a ScaLAPACK driver routine.

Most of the linear algebra algorithms perform a succession of elementary transformations on rows or columns of a matrix. Therefore, most of the communication operations performed within a ScaLAPACK routine involve process rows or columns. If one assumes that there is roughly the same number of of communication operations in both dimensions of the process grid, then a square two-dimensional process grid clearly offers the greatest scope for the parallelization of the communication operations.

Nevertheless, a few exceptions to this rule exist. First, if the user's interconnection network physically supports only one processor communicating at a time (e.g., ethernet), then better performance will be achieved on a one-dimensional process grid. Indeed, the smaller number of larger messages to be exchanged on a one-dimensional grid prevents the competition for resources from becoming a critical performance factor. Second, for a small number (say, eight) of processes, it is often slightly preferable to select a one-dimensional process grid—simply because there are not enough processes to make a large difference.

Most of the computation in the ScaLAPACK routines is performed in a blocked fashion by using Level 3 BLAS, as is done in LAPACK. The logical computational blocking factor used within the Level 3 PBLAS may differ from the distribution block size. Consequently, the performance of the ScaLAPACK library is not very sensitive to the physical distribution block size, as long as the extreme case is avoided. Very large distribution blocking factors do lead to computational imbalance. The chosen logical block size affects the amount of workspace needed on every process. This amount of workspace is typically large enough to contain a logical block of rows or columns of the distributed matrix operand. Therefore, the larger the logical block size, the greater the necessary workspace or, put another way, the smaller the problem that can be solved on a given grid of processes. For Level 3 BLAS block-partitioned algorithms, one dimension of the matrix operands is locally equal to the logical block size. Therefore, it is good practice to choose the logical block size to be the problem size for which the BLAS matrix-multiply routine achieves approximately 90% of its peak performance.

2.4 Installation

If the components of the ScaLAPACK library are properly installed, one should obtain performance results that are consistent with the computation and communication capabilities of the application computer. It is therefore a recommended practice to check the quality of the installation of the ScaLAPACK components as well as the performance of the hardware. Testing and timing programs are provided with all of the ScaLAPACK software components to diagnose installation problems. The above selection method of the distribution parameters generally achieves within 25% of the best possible performance. A rapid performance evaluation can be made at this stage to verify that the performance results obtained are reasonable and coherent. Finer tuning of these parameters can, of course, improve performance further.

3 Performance, Portability, and Scalability

How can we provide **portable** software for dense linear algebra computations that is **efficient** on a wide range of modern distributed-memory concurrent computers? Answering this question—and providing the appropriate software—has been an objective of the ScaLAPACK project.

The ScaLAPACK software has been designed specifically to achieve high efficiency for a wide range of modern distributed-memory concurrent computers. Examples of such machines include the Cray T3D and T3E, the IBM Scalable POWERparallel SP series, the Intel iPSC and Paragon, the nCube-2/3, networks and clusters of workstations (NoWs and CoWs), and "piles" of PCs (PoPCs).

For clarity of discussion, we consider this large diversity of architectures under the single model logical distributed-memory computer representation. This model consists of p processors that are connected by a message-passing interconnection network. Each processor has its own memory, called the local memory, which is accessible only to that processor. The time to access remote memory is longer than the time to access local memory. Such a computer is often referred to as a Non-Uniform Memory Access (NUMA) machine.

For the sake of simplicity, we also assume that all processors can be treated equally in terms of local performance and that the communication rate between two processors is independent from the processors considered. The local processor performance and the network performance and connectivity are therefore the main machine factors affecting the performance achieved by the ScaLAPACK drivers.

3.1 The BLAS as the Key to (Trans)portable Efficiency

The total number of floating-point operations performed by most of the ScaLAPACK driver routines for dense matrices can be approximated by the quantity $C_f N^3$, where C_f is a constant and N is the order of the largest matrix operand. For solving linear equations or linear least squares, C_f is a constant depending solely on the selected algorithm. The algorithms used to find eigenvalues and singular values are iterative; hence, for these operations, the constant C_f truly depends on the input data as well. It is, however, customary or "standard" to consider the values of the constants C_f for a fixed number of iterations. The "standard" constants C_f range from 1/3 to 27, as shown in Table 4.

The performance of the ScaLAPACK drivers is thus bounded above by the performance of a computation that could be partitioned into p independent chunks of $C_f N^3/p$ flops each. This upper bound is referred to hereafter as the *peak performance* and can be computed as the product of $C_f N^3/p$ and the highest reachable local processor flop rate. Hence, for a given problem size N and assuming a uniform distribution of the computational tasks, the most important factors determining the overall performance are the number p of processors involved in the computation and the local processor flop rate.

In a serial computational environment, *transportable efficiency* is the essential motivation for developing blocking strategies and block-partitioned algorithms [2, 3, 14, 28]. The linear algebra package (LAPACK) [3] is the archetype of such a strategy. The LAPACK software is constructed as much as possible out of calls to the BLAS. These kernels confine the impact of machine architecture differences within a small number of routines. The efficiency and portability of the LAPACK software are then achieved by combining native and efficient BLAS implementations with portable high-level components.

The BLAS are subdivided into three levels, each of which offers increased scope for exploiting parallelism. This subdivision corresponds to three different kinds of basic linear algebra operations:

- Level 1 BLAS [30]: for vector operations, such as $y \leftarrow \alpha x + y$;
- Level 2 BLAS [16]: for matrix-vector operations, such as $y \leftarrow \alpha A x + \beta y$;
- Level 3 BLAS [15]: for matrix-matrix operations, such as $C \leftarrow \alpha A B + \beta C$.

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

The performance potential of the three levels of BLAS is strongly related to the ratio of floating-point operations to memory references, as well as to the reuse of data when it is stored in the higher levels of the memory hierarchy. Consequently, the Level 1 BLAS cannot achieve high efficiency on most modern supercomputers. The Level 2 BLAS can achieve near-peak performance on many vector processors. On RISC microprocessors, however, their performance is limited by the memory access bandwidth bottleneck. The greatest scope for exploiting the highest levels of the memory hierarchy as well as other forms of parallelism is offered by the Level 3 BLAS [3].

The previous reasoning applies to distributed-memory computational environments in two ways. First, in order to achieve overall high performance, it is necessary to express the bulk of the computation local to each process in terms of Level 3 BLAS operations. Second, designing and developing a set of parallel BLAS (PBLAS) for distributed-memory concurrent computers should lead to an efficient and straightforward port of the LAPACK software. This is the path followed by the ScaLAPACK project [8, 19] as well as others [1, 7, 12, 21]. As part of the ScaLAPACK project, a set of PBLAS has been early designed and developed [11, 9].

3.2 Block Cyclic Data Layout as the Key to Load Balancing and Software Reuse

The way the data is distributed over the memory hierarchy of a computer is of fundamental importance to load balancing and software reuse. The block cyclic data layout allows a reduction of the overhead due to load imbalance and data movement. Block-partitioned algorithms are used to maximize the local processor performance.

Since the data decomposition largely determines the performance and scalability of a concurrent algorithm, a great deal of research [10, 22, 24, 26] has focused on different data decompositions [4, 6, 27]. In particular, the two-dimensional block cyclic distribution [29] has been suggested as a possible general-purpose basic decomposition for parallel dense linear algebra libraries [13, 25, 31], such as ScaLAPACK.

Block cyclic distribution is beneficial because of its scalability [18], load balance, and communication [25] properties. The block-partitioned computation then proceeds in consecutive order just like a conventional serial algorithm. This essential property of the block cyclic data layout explains why the ScaLAPACK design has been able to reuse the numerical and software expertise of the sequential LAPACK library.

3.3 The BLACS as an Efficient, Portable, and Adequate Message-Passing Interface

The total volume of data communicated by most of the ScaLAPACK driver routines for dense matrices can be approximated by the quantity $C_v\, N^2$, where N is the order of the largest matrix operand. The number of messages, however, is proportional to N and can be approximated by the quantity $C_m\, N/nb$, where nb is the logical blocking factor used in the computation. Similar to the situation described above, the "standard" constants C_v for the communication volume depend upon the performed computation and are of the same order as the floating-point-operation constants C_f shown in Table 4.

Developing an adequate message-passing interface specialized for linear algebra operations has been one of the first achievements of the ScaLAPACK project. The Basic Linear Algebra Communications Subprograms (BLACS) [20] were thus specifically designed to facilitate the expression of the relevant communication operations. The simplicity of the BLACS interface, as well as the rigor of their specification, allows for an easy port of the entire ScaLAPACK software. Currently, the BLACS have been efficiently ported on machine-specific message-passing libraries such as MPL (IBM) and NX (Intel), as well as more generic interfaces such as PVM and MPI. The BLACS overhead has been shown to be negligible [20].

The specificity and limited scope of the BLACS significantly contribute to its ease of use, portability, and efficiency. In addition, the BLACS interface provides the user and library designer with an appropriate level of notation. Indeed, the BLACS operate on typed two-dimensional arrays. 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, or to perform global reductions (sums, maxima and minima). Also included are routines to establish, modify, or query the process grid. The BLACS provide an adequate interface level for linear algebra communication operations.

For ease of use and flexibility, the BLACS send operation is **locally blocking**; that is, the return from the send operation indicates that the resources may be reused. However, since this feature depends only on local information, it is unknown whether the receive operation has been called. Buffering is necessary on the sending or the receiving process. The BLACS receive operation is **globally blocking**. The Return from the receive operation indicates that the message has been (sent and) received. On systems natively supporting globally blocking sends, nonblocking sends coupled with buffering are used to simulate locally blocking sends. This extra buffering operation may cause a slight performance degradation on those systems.

The BLACS broadcast and combine operations feature the ability to select different virtual network topologies. This easy-to-use built-in facility allows for the expression of various message scheduling approaches, such as a communication pipeline. This unique and distinctive BLACS characteristic is necessary for achieving the highest performance levels on distributed-memory platforms.

3.4 Parallel Efficiency

An important performance metric is *parallel efficiency*. Parallel efficiency, $E(N,p)$, for a problem of size N on p processors is defined in the usual way [22, 29] by

$$E(N,p) = \frac{1}{p}\frac{T_{\text{seq}}(N)}{T(N,p)},$$

where $T(N,p)$ is the runtime of the parallel algorithm, and $T_{\text{seq}}(N)$ is the runtime of the best sequential algorithm. For dense matrix computations, an implementation is said to be *scalable* if the parallel efficiency is an increasing function of N^2/p, the problem size per processor. The algorithms implemented in the ScaLAPACK library are scalable in this sense.

Figure 1 shows the scalability of the ScaLAPACK implementation of the LU factorization on the Intel XP/S MP Paragon. The figure shows the speed in Mflop/s of the ScaLAPACK LU factorization routine PDGETRF for different machine configurations. When the number of nodes is scaled by a constant factor (2 in the figure), the same efficiency or speed per node is achieved for equidistant problem sizes on a logarithmic scale. In other words, maintaining a constant memory use per node allows efficiency to be maintained. In practice, however, a slight degradation is acceptable. The ScaLAPACK driver routines in general feature the same scalability behavior up to a constant factor that depends on the exact number of floating-point operations and the total volume of data exchanged during the computation.

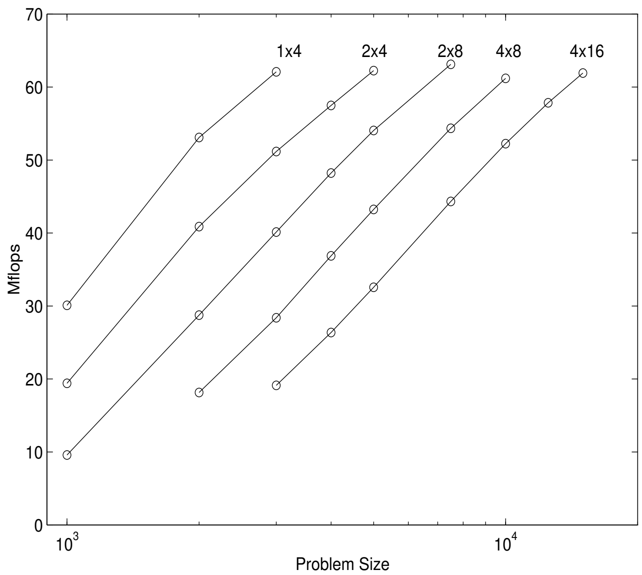

FIG. 1. *LU Performance per Intel Paragon node*

The *performance* of the algorithms implemented in ScaLAPACK is also measured in megaflop/s per second (or gigaflop/s per second). This measurement is appropriate for large dense linear algebra computations since the computation cost dominates the communication cost. In the following, the time to execute one floating-point operation by one processor is denoted t_f. The time to communicate a message between two processors is approximated by a linear function of the number of items communicated. The function is the sum of the time to prepare the message for transmission (t_m) and the time taken by the message to traverse the network to its destination, that is, the product of its length by the time to transfer one data item (t_v). Alternatively, t_m is also called the *latency,* since it is the time to communicate a message of zero length. On most modern interconnection networks, the order of magnitude of the latency varies between a microsecond and a millisecond.

The bandwidth of the network is also referred to as its *throughput.* It is proportional to the reciprocal of t_v. On modern networks, the order of magnitude of the bandwidth is a megabyte per second. For a scalable algorithm with N^2/p held fixed, one expects the performance to be proportional to p. The algorithms implemented in ScaLAPACK are scalable in this sense. It follows that the execution time of the ScaLAPACK drivers can thus be approximated by

$$T(N,p) = \frac{C_f\, N^3}{p}\, t_f + \frac{C_m\, N^2}{\sqrt{p}}\, t_v + \frac{C_m\, N}{nb}\, t_m \quad \text{i.e.,} \quad T_{seq}(N,p) = C_f\, N^3\, t_f.$$

The corresponding parallel efficiency can then be approximated by

$$E(N,p) = (1 + \frac{1}{nb}\frac{C_m\, t_m}{C_f\, t_f}\frac{p}{N^2} + \frac{C_v\, t_v}{C_f\, t_f}\frac{\sqrt{p}}{N})^{-1}. \tag{1}$$

Equation 1 illustrates, in particular, that the communication versus computation performance ratio of a distributed-memory concurrent computer significantly affects parallel efficiency. The ratio of the latency to the time per flop (t_m/t_f) greatly affects the parallel efficiency of small problems. The ratio of the network throughput to the flop rate (t_f/t_v) significantly affects the parallel efficiency of medium-sized problems. For large problems, the processor flop rate ($1/t_f$) is the dominant factor contributing to the parallel efficiency of the parallel algorithms implemented in ScaLAPACK.

4 ScaLAPACK Strategy

4.1 Software Hierarchy

The ScaLAPACK strategy for combining efficiency with portability is to construct the software as much as possible out of calls to the PBLAS for global computation. These routines in turn rely on the BLAS for local computation and on the BLACS for communication.

The efficiency of the ScaLAPACK software depends on efficient implementations of the BLAS and the BLACS being provided by computer vendors (or others) for their machines. Thus, the BLAS and the BLACS form a low-level interface between ScaLAPACK software and different machine architectures. Above this level, all of the ScaLAPACK software is portable.

In this article, performance results are presented for three different distributed-memory concurrent computers: the IBM Scalable POWERparallel 2, the Intel XP/S Paragon, and a network of SPARC Ultra 1's connected via switched ATM. Table 1 summarizes the relevant technical features of these machines.

For each machine this table shows the type of processor node, the peak flop rate per node, the peak latency, the bandwidth of the interconnection network, and the amount of physical memory per node. The numbers in parentheses are the relevant and corresponding numbers that a user program can achieve. For example, the flop rate in parentheses is the flop rate of the BLAS matrix-multiply. The latency and bandwidth are the corresponding values achieved by the BLACS. Finally, the amount of memory per node in parentheses is an approximation of the largest amount of memory available to the user's program.

TABLE 1
Machine Characteristics

Machine	Node Type	Flop Rate (Mflop/s)	Latency (μs)	Bandwidth (MB/s)	Memory per Node (MB)
IBM SP2	Thin	266 (200)	50 (400)	40 (30)	128 (100)
Intel Paragon	MP	100 (90)	29 (60)	175 (70)	64 (52)
NoW	SPARC Ultra 2		(450)	(10)	

The performance numbers presented have been obtained on real double precision data on all the computers. This corresponds to 64-bit floating-point arithmetic on all machines tested.

4.2 Performance of Selected PBLAS Routines

On a distributed-memory concurrent computer consisting of RISC processors, such as the IBM SP2 or the Intel MP Paragon, the performance of the Level 2 PBLAS is limited by the rate of data movement between different levels of memory within a processor. In other words, the performance of the Level 2 BLAS on each processor considerably limits the performance of the equivalent distributed operation. Table 2 shows the performance results obtained by the general matrix-vector multiply PBLAS routine PDGEMV.

TABLE 2
Speed in Megaflop/s of the PBLAS Matrix-Vector Multiply Routines for matrices of order N with TRANS='N' PDGEMV

	Processor Grid	Block Size	Values of N				
			2000	4000	6000	8000	10000
IBM SP2	2×2	50	177	185	187		
	4×4	50	550	683	678	639	710
	8×8	50	1310	2073	1378	1754	2455
Intel MP Paragon	2×2	32	162				
	4×4	32	543	620	666		
	8×8	32	1597	2117	2356	2461	2563
NoW SPARC Ultra ATM	2×2	64	117	126			
	2×4	64	218	241			
	3×4	64	285	345			

This limitation is overcome by the Level 3 PBLAS, which locally perform $O(\frac{N^3}{p\sqrt{p}})$ floating-point operations on $O(\frac{N^2}{p})$ data. The flop rate achieved by every processor for such a distributed operation is then much higher. Table 3 shows the performance results obtained by the general matrix-matrix multiply PBLAS routine PDGEMM for square matrices of order N.

TABLE 3

Speed in Megaflop/s of the PBLAS Matrix-Matrix Multiply Routines PDGEMM for matrices of order N with TRANSA='N' and TRANSB='N'

	Processor Grid	Block Size	Values of N 2000	4000	6000	8000	10000
IBM SP2	2×2	50	755				
	4×4	50	2514	2850	3040		
	8×8	50	6205	8709	9862	10468	10774
Intel MP Paragon	2×2	32	330				
	4×4	32	1233	1281	1334		
	8×8	32	4496	4864	5030	5103	5257
NoW SPARC Ultra ATM	2×2	64	218	340			
	2×4	64	529	734	845		
	2×6	64	510	956	1146	1205	

5 Solution of Common Numerical Linear Algebra Problems

This section contains performance numbers for selected driver routines. These routines provide complete solutions for the most common problems of numerical linear algebra and are the routines users are most likely to call:

- Solve an N by N general system of linear equations with one right-hand side using PDGESV.
- Solve an N by N symmetric positive definite system of linear equations with one right-hand side using PDPOSV.
- Solve an N by N linear least squares problem with one right-hand side using PDGELS. item Find the eigenvalues and eigenvectors of an N by N

Data is provided for a variety of distributed-memory concurrent computers. All timings were obtained by using the machine-specific optimized BLAS available on each machine. For the IBM Scalable POWERparallel 2, the ESSL BLAS were used. In all cases the data consisted of 64-bit floating-point numbers. For each machine and each driver, a range of problems was run on different number of processors. Different physical distribution block sizes were tried, with data for the fastest run reported in the tables below. Similarly, whenever applicable, UPLO='L' and UPLO='U' were timed, but times are reported only for UPLO='U'. The test matrices were generated with randomly distributed entries. All run times are reported in seconds, and block size is denoted by nb. The value of the physical distribution block size as well as the process grid shape was chosen to make $N = 2000$ optimal. It is not necessarily the best choice for the entire range of problem sizes.

Table 4 presents "standard" floating-point operation counts for ScaLAPACK drivers.

5.1 Factorizations for Solving Linear Equations

The LU and Cholesky factorizations are the simplest block algorithms to derive for the block cyclic layout. Table 5 illustrates the speed of the ScaLAPACK routine for the LU factorization of a real matrix, PDGETRF. This corresponds to 64-bit floating-point arithmetic on all machines tested. The distribution block size is also used as the partitioning

TABLE 4

"Standard" Floating-Point Operation Counts for Some ScaLAPACK Drivers for N by N Matrices

Driver	Options	Operation Count
PxGESV	1 Right-hand Side	$2/3 \cdot N^3$
PxPOSV	1 Right-hand Side	$1/3 \cdot N^3$
PxGELS	1 Right-hand Side	$4/3 \cdot N^3$
PxSYEVX	Eigenvalues Only	$4/3 \cdot N^3$
PxSYEVX	Eigenvalues and Eigenvectors	$9 \cdot N^3$

unit for the computation and communication phases. Table 6 gives similar results for the Cholesky factorization.

TABLE 5

Speed in Megaflop/s of PDGETRF for Square Matrices of Order N

	Processor Grid	Block Size	Values of N: 2000	5000	7500	10000	15000
IBM SP2	1×4	50	426	606			
	2×8	50	767	1574	1925	2165	
	4×16	50	1026	3182	4433	5730	7151
Intel MP Paragon	1×4	32	215	283			
	2×8	32	479	876	1016		
	4×16	32	768	2147	2888	3383	3988
NoW SPARC Ultra ATM	2×2	64	193	336			
	2×4	64	193	491			
	2×6	64	207	611		939	

The right-looking variants of the LU and Cholesky factorizations were chosen for ScaLAPACK because they minimize the total communication volume, that is, the aggregated amount of data transferred between processors during the operation.

ScaLAPACK provides LU and Cholesky factorizations for band matrices. For small bandwidth, divide-and-conquer algorithms have been chosen despite their higher cost in terms of floating-point operations. A more detailed performance analysis can be found in [5].

5.2 QR Factorization

The traditional algorithm for QR factorization is based on the use of elementary Householder matrices of the general form

$$H = I - \tau v v^T,$$

where v is a column vector and τ is a scalar. This leads to an algorithm with very good vector performance, especially if coded to use Level 2 PBLAS.

The key to developing a distributed block form of this algorithm is to represent a product of b elementary Householder matrices of order n as a block form of a Householder

Table 6

Speed in Megaflop/s of PDPOTRF for Matrices of Order N with UPLO='U'

	Processor Grid	Block Size	Values of N				
			2000	5000	7500	10000	15000
IBM SP2	2×2	50	471	620			
	4×4	50	1106	1830	2129	2323	
	8×8	50	1891	4549	5830	6893	8127
Intel MP Paragon	2×2	32	197	258			
	4×4	32	514	830	949	970	
	8×8	32	1170	2310	2865	3246	3743
NoW SPARC Ultra ATM	2×2	64	177	347			
	2×4	64	137	408			
	3×4	64	114	379		765	

matrix . This can be done in various ways. ScaLAPACK uses the following form [32]:

$$H_1 H_2 \ldots H_b = I - VTV^T,$$

where V is an n by b matrix whose columns are the individual vectors $v_1, v_2, \ldots, v_b$ associated with the Householder matrices $H_1, H_2, \ldots, H_b$, and T is an upper triangular matrix of order b. Extra work is required to compute the elements of T, but once again this is compensated for by the greater speed of applying the block form. Table 7 summarizes results obtained with the ScaLAPACK routine PDGEQRF.

Table 7

Speed in Megaflop/s of PDGEQRF for Square Matrices of Order N

	Processor Grid	Block Size	Values of N				
			2000	5000	7500	10000	15000
IBM SP2	1×4	50	387	594			
Intel MP Paragon	1×4	32	201	255			
	2×8	32	528	825	898	930	
	4×16	32	1004	2354	2937	3263	3598
NoW SPARC Ultra ATM	2×2	64	221	352			
	2×4	64	219	476			
	2×6	64	228	613			

6 Conclusions

This article has presented some performance figures for ScaLAPACK routines. The figures are provided for illustration only and should not be regarded as a definitive up-to-date statement of performance. They have been selected from performance figures obtained in 1995–1996 during the development of version 1.4 of ScaLAPACK. All reported timings were obtained by using the optimized version of the BLAS available on each machine. For the IBM computers, the ESSL BLAS were used. The PVM and MPI versions of the BLACS was used for timings involving clusters of workstations; the BLACS written on top of MPL was used for the timings on the IBM SP-2; the BLACS written on top of NX was used for timings on the Intel Paragon.

Performance is affected by many factors that may change from time to time, such as details of hardware (cycle time, cache size), communication latency, bandwidth, compiler, and BLAS. To obtain up-to-date performance figures, one should use the timing programs provided with ScaLAPACK.

ScaLAPACK is portable across a wide range of distributed-memory environments such as the IBM SP series, Intel series (Gamma, Delta, Paragon), Cray T3 series, TM CM-5, clusters of workstations, and any system for which PVM [23] or MPI [33] is available. Similar to the BLAS and LAPACK, many of the goals of the ScaLAPACK project—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 machine dependencies to two standard libraries: the BLAS (Basic Linear Algebra Subroutines) and the BLACS (Basic Linear Algebra Communication Subroutines). ScaLAPACK will run on any machine where both the BLAS and the BLACS are available.

All ScaLAPACK-related software is publicly available on *netlib* via the URL

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

References

[1] M. Aboelaze, N. Chrisochoides, and E. Houstis, *The Parallelization of Level 2 and 3 BLAS Operations on Distributed Memory Machines*, Tech. Rep. CSD-TR-91-007, Purdue University, West Lafayette, IN, 1991.

[2] R. Agarwal, F. Gustavson, and M. Zubair, *Improving Performance of Linear Algebra Algorithms for Dense Matrices Using Algorithmic Prefetching*, IBM J. Res. Dev., 38 (1994), pp. 265–275.

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

[4] C. Ashcraft, *The Distributed Solution of Linear Systems Using the Torus-wrap Data mapping*, Tech. Rep. ECA-TR-147, Boeing Computer Services, Seattle, WA, 1990.

[5] L. S. Blackford, J. Choi, A. Cleary, J. Demmel, I. Dhillona, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, D. Walker, and R. C. Whaley, *" ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers - Design Issues and Performance "*, in Proceedings of the Supercomputer 96 Conference, IEEE Computer Society Press, November 1996.

[6] R. Brent, *The LINPACK Benchmark on the AP 1000*, in Frontiers, 1992, McLean, VA, 1992, pp. 128–135.

[7] R. Brent and P. Strazdins, *Implementation of BLAS Level 3 and LINPACK Benchmark on the AP1000*, Fujitsu Scientific and Technical Journal, 5 (1993), pp. 61–70.

[8] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers - Design Issues and Performance*, Computer Physics Communications, 97 (1996), pp. 1–15. (also LAPACK Working Note #95).

[9] 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*, LAPACK Working Note #100 Technical report UT CS–95–292, University of Tennessee, 1995.

[10] J. Choi, J. Dongarra, R. Pozo, and D. Walker, *"ScaLAPACK: A Scalable Linear Algebra Library for Distributed Memory Concurrent Computers"*, Tech. Rep. UT CS-92-181, LAPACK Working Note #55, University of Tennessee, 1992.

[11] J. Choi, J. Dongarra, and D. Walker, *PB-BLAS: A Set of Parallel Block Basic Linear Algebra Subroutines*, Concurrency: Practice and Experience, 8 (1996), pp. 517–535.

[12] A. Chtchelkanova, J. Gunnels, G. Morrow, J. Overfelt, and R. van de Geijn, *Parallel Implementation of BLAS: General Techniques for Level 3 BLAS*, Tech. Rep. TR95-49, Department of Computer Sciences, UT-Austin, 1995. Submitted to Concurrency: Practice and Experience.

[13] E. Chu and A. George, *QR Factorization of a Dense Matrix on a Hypercube Multiprocessor*, SIAM Journal on Scientific and Statistical Computing, 11 (1990), pp. 990–1028.

[14] M. Dayde, I. Duff, and A. Petitet, *A Parallel Block Implementation of Level 3 BLAS for MIMD Vector Processors*, ACM Trans. Math. Softw., 20 (1994), pp. 178–193.

[15] J. Dongarra, J. D. Croz, I. Duff, and S. Hammarling, *"A Set of Level 3 Basic Linear Algebra Subprograms"*, ACM Trans. Math. Softw., 16 (1990), pp. 1–28.

[16] J. Dongarra, J. D. Croz, S. Hammarling, and R. Hanson, *"An Extended Set of Fortran Basic Linear Algebra Subprograms"*, ACM Trans. Math. Softw., 14 (1988), pp. 1–32.

[17] J. Dongarra and R. van de Geijn, *"Two dimensional Basic Linear Algebra Communication Subprograms"*, Tech. Rep. UT CS-91-138, LAPACK Working Note #37, University of Tennessee, 1991.

[18] J. Dongarra, R. van de Geijn, and D. Walker, *"A Look at Scalable Dense Linear Algebra Librairies"*, Tech. Rep. UT CS-92-155, LAPACK Working Note #43, University of Tennessee, 1992.

[19] J. Dongarra and D. Walker, *Software Libraries for Linear Algebra Computations on High Performance Computers*, SIAM Review, 37 (1995), pp. 151–180.

[20] J. Dongarra and R. C. Whaley, *"A User's Guide to the BLACS v1.0"*, Tech. Rep. UT CS-95-281, LAPACK Working Note #94, University of Tennessee, 1995.

[21] R. Falgout, A. Skjellum, S. Smith, and C. Still, *The Multicomputer Toolbox Approach to Concurrent BLAS and LACS*, in Proceedings of the Scalable High Performance Computing Conference SHPCC-92, IEEE Computer Society Press, 1992.

[22] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, *"Solving Problems on Concurrent Processors"*, vol. 1, Prentice Hall, Englewood Cliffs, N.J, 1988.

[23] 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*, The MIT Press Cambridge, Massachusetts, 1994.

[24] G. Geist and C. Romine, *LU Factorization Algorithms on Distributed Memory Multiprocessor Architectures*, SIAM Journal on Scientific and Statistical Computing, 9 (1988), pp. 639–649.

[25] B. Hendrickson and D. Womble, *The Torus–wrap Mapping for Dense Matrix Calculations on Massively Parallel Computers*, SIAM Journal on Scientific and Statistical Computing, 15 (1994), pp. 1201–1226.

[26] G. Henry and R. van de Geijn, *Parallelizing the QR Algorithm for the Unsymmetric Algebraic Eigenvalue problem: Myths and Reality*, Tech. Rep. UT CS-94-244, LAPACK Working Note #79, University of Tennessee, 1994.

[27] S. Huss-Lederman, E. Jacobson, A. Tsao, and G. Zhang, *Matrix Multiplication on the Intel Touchstone DELTA*, Concurrency: Practice and Experience, 6 (1994), pp. 571–594.

[28] B. Kagström, P. Ling, and C. van Loan, *GEMM-Based Level 3 BLAS: High-Performance Model Implementations and Performance Evaluation Benchmark*, Tech. Rep. UMINF 95-18, Department of Computing Science, Umea University, 1995. Submitted to ACM TOMS.

[29] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing*, The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1994.

[30] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh, *"Basic Linear Algebra Subprograms for Fortran Usage"*, ACM Trans. Math. Softw., 5 (1979), pp. 308–323.

[31] W. Lichtenstein and S. L. Johnsson, *Block-Cyclic Dense Linear Algebra*, SIAM Journal on Scientific and Statistical Computing, 14 (1993), pp. 1259–1288.

[32] R. Schreiber and C. Van Loan, *A storage efficient WY representation for products of Householder transformations*, SIAM J. Sci. Stat. Comput., 10 (1989), pp. 53–57.
[33] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. J. Dongarra, *MPI: The Complete Reference*, MIT Press, Cambridge, Massachusetts, 1996.