CRPC RESEARCH INTO LINEAR ALGEBRA SOFTWARE FOR HIGH PERFORMANCE COMPUTERS

Jaeyoung Choi[1]
Jack J. Dongarra[2]
Roldan Pozo[3]
Danny C. Sorensen[4]
David W. Walker[1]

[1]MATHEMATICAL SCIENCES SECTION, OAK RIDGE NATIONAL LABORATORY, OAK RIDGE, TENNESSEE
[2]DEPARTMENT OF COMPUTER SCIENCE, UNIVERSITY OF TENNESSEE, KNOXVILLE, TENNESSEE AND MATHEMATICAL SCIENCES SECTION, OAK RIDGE NATIONAL LABORATORY, OAK RIDGE, TENNESSEE
[3]DEPARTMENT OF COMPUTER SCIENCE, UNIVERSITY OF TENNESSEE, KNOXVILLE, TENNESSEE
[4]DEPARTMENT OF COMPUTATIONAL AND APPLIED MATHEMATICS, RICE UNIVERSITY, HOUSTON, TEXAS

Summary

In this paper we look at a number of approaches being investigated in the Center for Research on Parallel Computation (CRPC) to develop linear algebra software for high-performance computers. These approaches are exemplified by the LAPACK, templates, and ARPACK projects. LAPACK is a software library for performing dense and banded linear algebra computations, and was designed to run efficiently on high-performance computers. We focus on the design of the distributed-memory version of LAPACK, and on an object-oriented interface to LAPACK.

The International Journal of Supercomputer Applications, Volume 8, No. 2, Summer 1994, pp. 99–118
© 1994 Massachusetts Institute of Technology.

1. Introduction

Linear algebra lies at the heart of the research program of the Center for Research on Parallel Computation (CRPC). It provides critical underpinning for much of the work on higher-level optimization algorithms and the numerical solution of partial differential equations. It has proved to be a rich source of basic problems for work on compiler management of memory hierarchies and compiling for distributed-memory machines. Finally, it is serving as a testbed for our ideas on designing, building, and distributing libraries of scalable mathematical software.

In this paper we present a survey of our research into the design, development, and use of software libraries for performing linear algebra computations on distributed-memory concurrent computers. This research is being conducted under CRPC auspices by collaborating groups at Rice University, the University of Tennessee at Knoxville, and Oak Ridge National Laboratory.

In Section 2, we begin by looking at our framework for library design. The rest of the paper focuses on three main topics: LAPACK, templates, and ARPACK. Section 3 presents key ideas in the design of LAPACK, a widely used software library for performing dense and banded matrix computations on high-performance computers. The design and performance of a distributed-memory version of the LAPACK software library are discussed, and an object-oriented interface to this package is described. In Section 4, we present an approach to the solution of sparse linear algebra problems with reusable software "templates" or code that describes the essential features of an algorithm and that can also be customized by users for particular applications. The ARPACK software package for performing large-scale eigenanalysis is described in Section 5 and is based on a new variant of the Arnoldi process. Concluding remarks and directions for future work close the article proper. An appendix explains how to obtain the software described in this paper using *netlib* and *xnetlib*.

"Linear algebra provides critical underpinning for work on higher-level optimization algorithms and the numerical solution of partial differential equations. It has proved to be a rich source of basic problems for work on compiler management of memory hierarchies and compiling for distributed-memory machines."

2. Framework for Library Development on Advanced Architectures

The design and implementation of a library of scalable mathematical software is critical to the success of our efforts to make parallel computers truly useful to scientists and engineers. We are building a prototype library of scalable methods for solving problems in numerical linear algebra.

We are investigating three approaches in the design of such a library.

1. The first approach is to take existing well-tested software, in this case LAPACK, and introduce small modifications in order to develop distributed memory implementations. In this way, we can quickly develop a core set of routines, leveraging existing software as much as possible. This approach is described in Section 3.
2. The second approach is a long-term algorithmic research project aimed at developing optimal strategies tailored to various parallel architectures. This will include both SIMD and MIMD machines. This may require abandoning our existing algorithm and software base and the development of completely new approaches. The research into software templates and ARPACK, described in Sections 4 and 5 respectively, are examples of this type of research project.
3. The third approach is to take High Performance Fortran (HPF) as a base language and develop a library that relies on the compiler to perform much of the machine mapping. Here, the design goal is to construct algorithms that are blockable and distributable across a variety of architectures and to encode enough information into the HPF specification to allow translation into efficient code.

By pursuing all three approaches, we expect to gain a deep understanding of the issues involved in designing and building libraries of scalable mathematical software, while exercising the HPF technology developed within CRPC. As a goal, we hope to produce libraries that provide a clean, architecture-independent interface to the end user.

This paper will only be concerned with the first two of these approaches. We are currently considering the development of an HPF version of LAPACK that will make use of the compiler's knowledge of data distribution, and thereby relieve the user of the need to explicitly pass data distribution information into each LAPACK routine. This work is ongoing, and will be described elsewhere in the future.

3. LAPACK on Concurrent Supercomputers

The LAPACK project started in 1987 in an effort to design a linear algebra library for conventional supercomputers and high-performance workstations. ScaLAPACK was a follow-on project whose original aim was to provide for users of MIMD distributed-memory concurrent computers the same functionality as that provided by LAPACK for shared-memory architectures. We now believe that it is possible to make ScaLAPACK fully compatible with LAPACK, so that the calling syntax of both libraries is identical, thereby allowing transparent migration of applications between shared-memory and distributed-memory architectures. Thus, the distinction between ScaLAPACK and LAPACK is essentially historical, with the former originally developed for distributed-memory machines, and the latter for shared-memory machines. When it is necessary to distinguish between the shared-memory and distributed-memory versions we shall use the terms LAPACK-SM and LAPACK-DM, respectively; otherwise, we shall use LAPACK to refer generically to both versions.

The development of sparse matrix algorithms is an area of intense research activity, largely because of their importance in the numerical solution of partial differential equations. Dense matrix computations are less pervasive, but also have important applications, as discussed in a recent survey by Edelman (1993). A major source of large dense linear systems is the solution of problems by the boundary-element method. In this method integral equations defined on the boundary of a region of interest are used to compute some final desired quantity in three-dimensional space. The dense linear systems that are generated are commonly solved using LU factorization. Electromagnetic scattering studies make use of the boundary-element method, which is usually referred to as the *method of moments* in this context (Harrington, 1990; Wang, 1991). This approach is used in the design of aircraft with small radar cross-sections and in the design of satellite antennae. Boundary-element methods are also used in the study of fluid flows, and here the variant of the boundary-element method used is called the *panel method* (Hess, 1990; Hess and Smith, 1967).

3.1 LAPACK ON NUMA MACHINES

Modern supercomputers may be classified as nonuniform memory access (NUMA) machines. That is, they possess hierarchical memories in which different levels in the hierarchy are characterized by different access times. Registers are the upper level of memory, and here access time is least. Caches are an intermediate level in the memory hierarchy. In shared-memory machines the lowest level in the memory hierarchy is the shared memory. In distributed-memory machines each processor has its own local memory, so the aggregate off-processor memory of all other processors forms the lowest level of the memory hierarchy. LAPACK attains high performance on NUMA machines by maximizing the reuse of data in the upper levels of memory, so that time-consuming accesses to lower levels are minimized. This is done using a two-fold approach. First, the frequency of data movement between the lower and intermediate levels of memory is controlled by recasting the numerical algorithms in block-partitioned form. This approach ensures that memory accesses are localized ("locality of reference") and can be deferred so that data are moved between the lower and intermediate levels in blocks. On shared-memory machines the use of block-partitioned algorithms minimizes the frequency of data movement between shared memory and cache, while on distributed-memory machines it reduces the frequency of communication between processors. Second, the movement of data between the intermediate and upper levels of the memory hierarchy is controlled by means of optimized assembly code for the most heavily used, compute-intensive parts of an algorithm. Fortunately, in the block-partitioned algorithms used in

LAPACK these correspond to Level 3 BLAS routines, optimized assembly code versions of which exist for the processors comprising most modern supercomputers.

Maximizing data reuse in the upper levels of memory through the use of block-partitioned algorithms is the cornerstone of the successful implementation of LAPACK on shared-memory and distributed-memory supercomputers. Optimized, concurrent Level 2 and Level 3 BLAS routines are used as building blocks for the LAPACK routines. The approach taken to parallelizing concurrent BLAS routines differs for shared-memory and distributed-memory machines. On shared-memory machines, the assignment of work to processors is determined by the compiler, typically by parallelizing the outermost loop(s) over blocks. The innermost loops are written in assembly code. One of the aims of High Performance Fortran (HPF) is to provide a similar level of compiler support for distributed-memory machines. However, in developing distributed-memory versions of the Level 2 and Level 3 BLAS we seek to optimize performance by manually parallelizing the appropriate loops. In addition, the data distribution is specified through subroutine calls, rather than by data distribution directives, as in HPF.

3.2 DISTRIBUTED MATRICES

In many linear algebra algorithms the distribution of work may become uneven as the algorithm progresses, for example as in LU factorization, in which rows and columns are successively eliminated from the computation. LAPACK-DM, therefore, makes use of the block-cyclic data distribution in which matrix blocks separated by a fixed stride in the row and column directions are assigned to the same processor. A number of researchers have made use of the block-cyclic data distribution in parallel dense linear algebra algorithms (Choi et al., 1992; Choi, Dongarra, and Walker, 1993a; Dongarra and Ostrouchov, 1990; Dongarra, van de Geijn, and Walker, 1992; Lichtenstein and Johnsson, 1993). A block-cyclic data distribution is parameterized by the four numbers P, Q, r, and c, where $P \times Q$ is the processor template and $r \times c$ is the block size. All LAPACK-DM routines work for arbitrary values of these parameters, subject to certain "compatibility conditions." Thus, for example, in the LU factorization routine we require that the blocks be square, since nonsquare blocks would lead to additional software complexity and communication overhead. When multiplying two matrices, $C = AB$, we require that all three matrices are distributed over the same $P \times Q$ process template; rectangular blocks are permitted, but we require that if the blocks of matrix A are $r \times t$, those of B and C must be $t \times c$ and $r \times c$, respectively, to enable us to multiply individual blocks of A and B to form blocks of C.

In the block-cyclic data distribution mapping of a global index, m, can be expressed as $m \mapsto (p,b,i)$, where p is the logical process number, b is the block number in process p, and i is the index within block b to which m is mapped. Thus, if the number of data objects in a block is r, the block-cyclic data distribution may be written

$$m \mapsto \left(\left\lfloor \frac{m \bmod T}{r} \right\rfloor, \left\lfloor \frac{m}{T} \right\rfloor, m \bmod r \right) \qquad (1)$$

where $T = rP$ and P is the number of processes. 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 processes, and another that distributes the columns over Q processes. It should be noted that Eq. (1) reverts to the cyclic distribution when $r = 1$, with local index $i = 0$ for all blocks. A block distribution is recovered when $r = \lceil M/P \rceil$, in which case there is a single block in each process with block number $b = 0$. Thus, we have

$$m \mapsto (m \bmod P, \lfloor m/P \rfloor, 0) \qquad (2)$$

for a cyclic data distribution, and

$$m \mapsto (\lfloor m/L \rfloor, 0, m \bmod L), \qquad (3)$$

for a block distribution, where $L = \lceil M/P \rceil$. A subtle distinction between the block distribution given by Eq. (3) and that often used elsewhere (see, for example, Fox et al., 1988; Van de Velde, 1990) should be noted. Consider the block distribution of six items over four processes. This is commonly distributed as (2,2,1,1), i.e., two items in two of the processes and one item in the other two processes. The block distribution given by Eq. (3) results in the distribution (2,2,2,0), so that one of the processes contains no data items. Clearly, since the load imbalance is measured by the difference between the maximum and the average loads, both distribution schemes have the same degree of load imbalance. We prefer the block distribution given by Eq. (3) because the arithmetic needed to convert between global and local indices is simpler, and because of the symmetry between the equations for the block and cyclic distributions (compare Eqs. (2) and (3)). There appear to be no other compelling reasons why one of these forms of block distribution should be preferred to the other in all cases.

The form of the block-cyclic data distribution given by Eq. (1) ensures that the block with global index 0 is placed in process 0, the next block is placed in process 1, and so on. However, it is sometimes necessary to offset the processes relative to the global block index so that, in general, the first block is placed in process p_0, the next in process $p_0 + 1$, and so on. For example, in the parallel, block LU factorization algorithm described by Dongarra et al. (1994), a rank-r update is applied to the unfactored portion of the matrix, E, in each step by multiplying a column of blocks, L_1, by a row of blocks, U_1, i.e., $E = E - L_1 U_1$. Here r is the block size. The three matrices involved in this update each have their (0,0) block in a different location of the process template. We, therefore, generalize the block-cyclic data distribution by replacing m on the right-hand side of Eq. (1) by $m' = m + rp_0$ to give

$$\begin{aligned} m \mapsto & \left(\left\lfloor \frac{m' \bmod T}{r} \right\rfloor, \left\lfloor \frac{m'}{T} \right\rfloor, m' \bmod r \right) \\ = & \left(\left(\left\lfloor \frac{m \bmod T}{r} \right\rfloor + p_0 \right) \bmod P, \right. \\ & \left. \left\lfloor \frac{m + rp_0}{T} \right\rfloor, m \bmod r \right). \end{aligned} \tag{4}$$

The inverse mapping is given by

$$(p, b, i) \mapsto Br + i = (p - p_0)r + bT + i, \tag{5}$$

where the global block number is given by $B = (p - p_0) + bP$.

The block-cyclic data distribution is the only data distribution supported by the LAPACK-DM routines,

"In many linear algebra algorithms the distribution of work may become uneven as the algorithm progresses. LAPACK-DM makes use of block-cyclic data distribution in which matrix blocks separated by a fixed stride in the row and column directions are assigned to the same processor."

and in its most general form is parameterized by P, Q, r, c, p_0, and q_0, where $P \times Q$ is the size of the process template, $r \times c$ is the block size, and (p_0, q_0) is the location in the template of the (0, 0) block of the matrix. The block-cyclic data distribution can reproduce most of the data distributions used in linear algebra computations. For example, one-dimensional distributions over rows or columns are obtained by choosing Q or P to be 1. Similarly, an $M \times N$ matrix can be decomposed into (nonscattered) blocks by choosing $r = \lceil M/P \rceil$ and $c = \lceil N/Q \rceil$. In algorithms, such as LU factorization, in which the distribution of work becomes uneven, a larger block size results in greater load imbalance, but reduces the frequency of communication between processors. There is, therefore, a tradeoff between load imbalance and communication startup cost, which can be controlled by varying the block size.

In addition to the load imbalance that arises as distributed data are eliminated from a computation, load imbalance may also arise due to computational "hot spots" when certain processes have more work to do between synchronization points than others. This is the case in the LU factorization algorithm, in which partial pivoting is performed over rows, and only a single column of the process template is involved in the pivot search while the other processes remain idle (Dongarra, van de Geijn, and Walker, 1994). Similarly, the evaluation of each block row of the U matrix requires the solution of a lower triangular system that involves only processes in a single row of the process template. The effect of this type of load imbalance can be minimized through the choice of P and Q. Another factor to be considered in choosing P and Q is the performance of collective communication routines, such as reduction and broadcast operations, that may be performed over the rows and columns of the process template.

3.3 BUILDING BLOCKS FOR LAPACK-DM

The LAPACK-DM routines are built out of a small number of modules. The most fundamental of these are the Basic Linear Algebra Communication Subprograms (BLACS) (Dongarra, 1991; Dongarra and van de Geijn, 1991), which perform common matrix-oriented communication tasks, and the sequential Basic Linear Algebra Subprograms (BLAS) (Dongarra et al., 1990; Dongarra et al., 1988; Lawson et al., 1979), in particular the Level 3 BLAS. LAPACK-DM can be ported with minimal code modification to any machine on which the BLACS and the BLAS are available. The distributed Level 3 BLAS (Choi, Dongarra, and Walker, 1993b, 1994b) and the Parallel Block BLAS (PB-BLAS), described in more detail below, are intermediate-level routines based on the BLACS and sequential BLAS. The BLACS, the sequential BLAS, the distributed Level 3 BLAS, and the PB-BLAS are the modules from which the higher-level LAPACK-DM routines are built. Thus, the entire LAPACK-DM package contains modules at a number of different levels. For many users the top-level LAPACK-DM routines will be sufficient to build applications. However, more expert users may make use of the lower-level routines to build customized routines that are not provided in LAPACK.

The BLACS package attempts to provide the same ease of use and portability for MIMD message-passing linear algebra communication that the BLAS provide for linear algebra computation. Therefore, future software for dense linear algebra on MIMD platforms could consist of calls to the BLAS for computation and calls to the BLACS for communication. Since both packages will have been optimized for each particular platform, good performance should be achieved with relatively little effort.

In the LAPACK-DM routines all interprocessor communication takes place within the distributed BLAS and the BLACS, so the source code of the top software layer of LAPACK-DM looks very similar to that of LAPACK-SM. The BLACS have been implemented for the Intel family of computers, the TMC CM-5, the CRAY T3D, the IBM SP-1, and for PVM.

The PB-BLAS are distributed Level 2 and 3 BLAS routines in which at least one of the matrix sizes is limited to the block size. That is, at least one of the matrices consists of a single row or column of blocks, and is located in a single row or column of the process template. An example of a PB-BLAS operation would be the multiplication of a matrix of $M \times N$ blocks by a "vector" of

N blocks. The PB-BLAS make use of calls to the sequential BLAS for local computations, and calls to the BLACS for communication. The PB-BLAS are used, for example, to perform block-oriented matrix/vector multiplications when reducing a column of blocks in the parallel Hessenberg reduction algorithm (Choi, Dongarra, and Walker, 1994a).

3.4 PERFORMANCE OF LAPACK-DM

The LAPACK-DM routines for performing LU, QR, and Cholesky factorizations of dense matrices have been extensively tested and benchmarked. Routines for reducing matrices to Hessenberg, tridiagonal, and bidiagonal form have also been developed. The main platform for testing these codes was a 128-node Intel iPSC/860 hypercube, although some of them have also been run on the Intel Delta and Paragon systems and the Thinking Machines CM-5. A PVM version of the LU factorization code has been run over a network of workstations. Details of the parallel factorization algorithms are given by Dongarra, van de Geijn, and Walker, (1994), together with performance results and models. A similar paper on the reduction routines is in preparation (Choi, Dongarra, and Walker, 1994a). Here we shall just present results from a small sample of the runs we have done on the Intel iPSC/860 hypercube to demonstrate the efficiency and scalability of the LAPACK-DM routines. Figure 1 shows isogranularity plots for the LU, QR, and Cholesky (LL^T) factorization routines. These plots show the variation in performance as a function of the number of processors, with the matrix size per processor kept fixed at 5 Mbytes/processor. If the factorization codes were perfectly scalable with respect to memory use these plots would all be linear. The fact that the curves deviate only mildly from linearity shows that the algorithms exhibit good scalability on the Intel iPSC/860, especially since less than half the memory available for applications is being utilized. Isogranularity plots for the Hessenberg, tridiagonal, and bidiagonal reduction routines are shown in Figure 2, again for a granularity of 5 Mbytes/processor. For the reduction routines the deviation from linearity appears

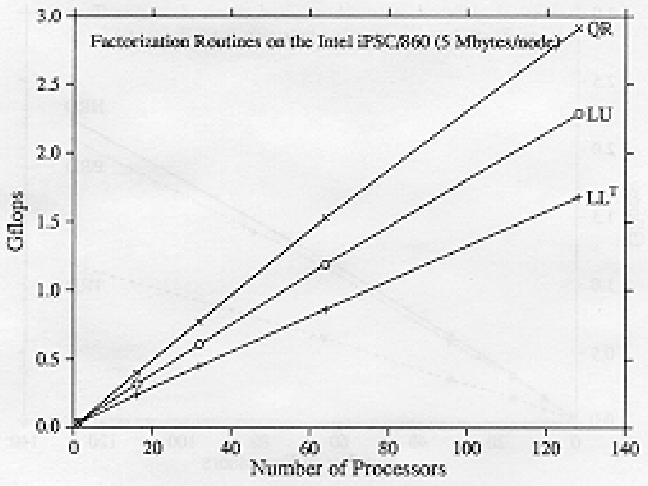

Fig. 1 Isogranularity plots for the dense LU, QR, and Cholesky factorization routines on the Intel iPSC/860 hypercube The matrix size per processor is fixed at 5 Mbytes.

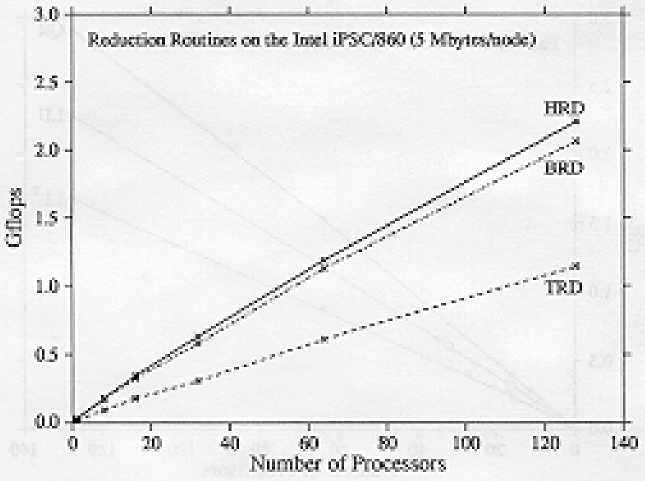

Fig. 2 Isogranularity plots for the Hessenberg (HRD), tridiagonal (TRD), and bidiagonal reduction (BRD) routines on the Intel iPSC/860 hypercube The matrix size per processor is fixed at 5 Mbytes.

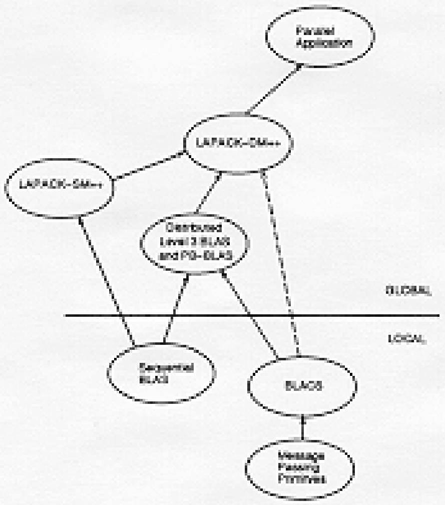

Fig. 3 Design Hierarchy of LAPACK-DM++ In an SPMD environment, components above the horizontal reference line represent a global viewpoint (a single distributed structure), while elements below represent a per-node local viewpoint of data.

a little more pronounced than for the factorization routines, but is still fairly small.

3.5 AN OBJECT-ORIENTED INTERFACE TO LAPACK

Our research activities in object-oriented extensions of LAPACK stems from two driving forces, one pulling, the other pushing. The pull comes from an increasing demand from the scientific computing community for C and C++ interfaces for numerical linear algebra libraries. A recent LAPACK survey revealed this to be one of the most common features users would like to see in future versions of LAPACK. The push stems from the ability of object-oriented software designs written in C++ to encapsulate naturally the complex data structures describing distributed matrices. These same mechanisms also help solve the "data distribution compatibility" problem often encountered when integrating distributed-memory libraries. LAPACK-DM++ solves this by exploiting the inheritance and dynamic-binding capabilities of C++. The result is that only *one* version of a library algorithm, such as the right-looking block LU factorization, need be maintained. This library code will work correctly with *any* matrix data distribution scheme.

Decoupling the matrix operations from the details of the data distribution provides two important benefits. First, it simplifies the description of a high-level algorithm and, second, it allows for postponement of data distribution until runtime. This is crucial if we wish to provide truly integrable numerical libraries since the appropriate matrix data distribution is strongly dependent on the way the data are utilized in other sections of the driving application.

In Figure 3 we illustrate the design hierarchy of LAPACK-DM++. A parallel SPMD application will utilize LAPACK-DM++ kernels to perform distributed linear algebra operations. Each node of a multicomputer runs a similar C++ program with calls to the LAPACK-DM++ interface. At this level distributed matrices are seen as a single object. The LAPACK-DM++ kernel, in turn, is built upon two important constituents: the basic algorithms of LAPACK++, and a parallel implementation of lower-level computational kernels (BLAS). Since the code parallelism is embedded in the low-level BLAS kernels, the driving routines that employ block matrix operations will look the same. Thus, the essential differences between LAPACK-SM++ and LAPACK-DM++ codes are simply in the declarations of the matrix objects that are supported.

The overhead introduced by the C++ interface is minimal because there is no extra data copying, nor is the function-call overhead significant, particularly when compared to the granularity of communication routines in message-passing architectures. For single-node performance, we have tested various prototype LAPACK++ (Dongarra, Pozo, and Walker, 1993) modules on several architectures and found that they achieve competitive performance with similar optimized Fortran LAPACK routines. Figure 4, for example, illustrates the performance of the LU factorization routine on an IBM RS/6000 Model 550 workstation. Three versions are compared: the native Fortran code with optimized BLAS, a C++ shell to this code, and the LU algorithm itself, which is written in C++ with optimized BLAS. This implementation used GNU g++ version 2.3.1 and the Level 3 BLAS routines from the native ESSL library. The performance results are nearly identical with those of optimized Fortran calling the same library.

LAPACK-DM++ includes support for the general, two-dimensional, block-cyclic matrix data distribution described in Section 3.2. However, its major advantage is that the library code for the LU algorithm will work correctly with *any* matrix data distribution. All we need to supply with each new matrix data distribution is a matching parallel BLAS library that can perform the basic functions. The key point here is that describing a new BLAS library is much simpler than specifying a new LAPACK library.

In short, the design of LAPACK-DM++ allows one to describe parallel, dense, linear algebra algorithms in terms of high-level primitives that are independent of the distribution of the matrices over the physical nodes of a multicomputer. Decoupling the matrix algorithm from a specific data distribution provides three important attributes: (1) it results in simpler code that more closely matches the underlying mathematical formulation, (2) it allows for one "universal" algorithm, rather than supporting one version for each data distribution needed, and (3) it allows one to postpone the data distribution decision until runtime.

4. Templates for Large Sparse Systems of Linear Equations

A new generation of even more massively parallel computers will soon emerge. Concurrent with the development of these more powerful parallel systems is a shift in the computing practices of many scientists and researchers. Increasingly, the tendency is to use a variety of distributed computing resources, with each individual task assigned to the most appropriate architecture, rather than to use a single, monolithic machine. The pace of these two developments—the emergence of highly parallel machines, and the move to a more highly distributed computing environment—has been so rapid that software developers have been unable to keep up. Part of the problem has been that supporting system software has inhibited this development. Consequently, exploiting the power of these technological advances has become more and more difficult. Much of the existing reusable scientific software, such as that found in commercial libraries and in public domain packages, is no longer adequate for the new architectures. If the full power of these new machines is to be realized, scalable libraries, comparable in scope and quality to those that currently exist, must be developed.

One of our goals as software designers is to communicate to the high-performance computing community algorithms and methods for the solution of systems of linear equations. In the past we have provided black-box software in the form of a mathematical software library, such as LAPACK, LINPACK, NAG, and IMSL. These software libraries provide for:

- easy interface with hidden details;
- reliability; the code should fail as rarely as possible;
- speed.

On the other hand, the high-performance computing community, which wants to solve complex, large-scale problems as quickly as possible, seems to want

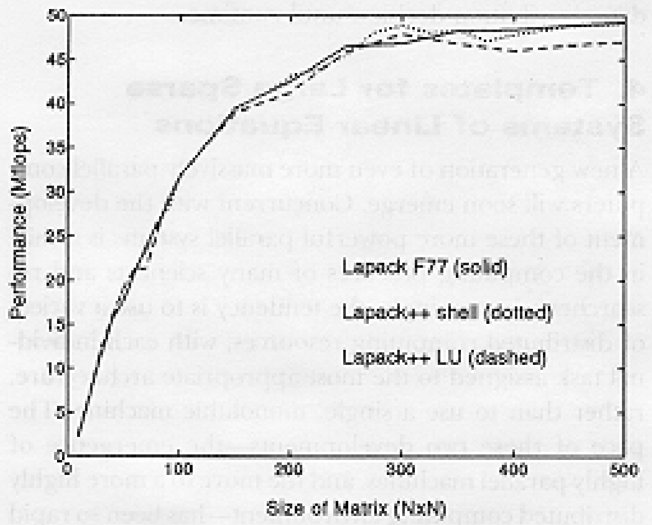

Fig. 4 Performance of LAPACK++ LU factorization on the IBM RS/6000 Model 550 workstation, using GNU g++ version 2.3.1 and BLAS routines from the IBM ESSL library Three versions are compared: the native Fortran code with optimized BLAS, a C++ shell to this code, and the LU algorithm itself, which is written in C++ with optimized BLAS.

- speed;
- access to internal details to tune data structures to the application;
- algorithms that are fast for the particular application even if not as reliable as general methods.

These differing priorities make for different approaches to algorithms and software. The first set of priorities has led us to produce "black boxes" for general problem classes. The second set of priorities seems to lead us to produce "template codes" or "toolboxes" where the users can assemble, modify, and tune building blocks starting from well-documented subparts. This leads to software that is not going to be reliable on all problems, and requires extensive user tuning to make it work. This is just what the black-box users do not want.

In scientific high-performance computing we see three different computational platforms emerging, each with a distinct set of users. The first group of computers contains the traditional supercomputer. Computers in this group exploit vector and modest parallel computing. They are general-purpose computers that can accommodate a large cross-section of applications while providing a high percentage of their peak computing rate. They are the computers typified by the CRAY Y-MP and C90, IBM ES/9000, and NEC SX-3—the so-called general-purpose vector supercomputers.

The second group of computers are the highly parallel computers. These machines often contain hundreds or even thousands of processors, usually RISC in design. The machines are usually loosely coupled, supplied with a switching network and having relatively long communication times compared with computation times. These computers are suitable for fine-grain and coarse-grain parallelism. As a system, the cost is usually less than the traditional supercomputer and the programming environment is very poor and primitive. There is no portability since users programs depend heavily on a particular architecture and on a particular software environment.

The third group of computers are the clusters of workstations. Each workstation usually contains a single very fast RISC processor. Each workstation is connected through a Local Area Network, or LAN, and so the communication time is very slow, making this setup not very suitable for fine-grain parallelism. This group usually has a rich software environment and operating system on a workstation node, usually UNIX. This solution is usually viewed as a very cost-effective solution compared to vector supercomputers and highly parallel computers.

Users are, in general, not a monolithic entity, but, in fact, represent a wide diversity of needs. Some are the sophisticated computational scientists who eagerly move to the newest architecture in search of ever higher performance. Others only wish to solve their problems with the least change in their computational approach.

We hope to satisfy the high-performance computing community's needs through the introduction of reusable software templates. With the templates we describe the basic features of the algorithms. These templates offer the opportunity for whatever degree of customization the user may desire, and also serve a valuable pedagogical role in teaching parallel programming and instilling a better understanding of the algorithms that are employed and results that are obtained. While providing reusable software templates we also hope to retain the delicate numerical details present in many algorithms.

We believe it is important for users to have trust in the algorithms, and hope this approach conveys the spirit of the algorithm and provides a clear path for implementation where the appropriate data structures can be integrated into the implementation. We believe that such an approach based on the use of templates will allow for easy modification to suit various needs. More specifically, each template should have:

- working software for matrices as general as the method allows;
- a mathematical description of the flow of the iteration;
- algorithms described in a Fortran-77 program with calls to BLAS (Dongarra et al., 1990; Dongarra et al., 1988; Lawson et al., 1979), and LAPACK routines (Anderson et al., 1992);
- discussion of convergence and stopping criteria;
- suggestions for extending a method to more specific matrix types (for example, banded systems);
- suggestions for tuning (for example, which preconditioners are applicable and which are not);
- performance: when to use a method and why;
- reliability: for what class of problems the method is appropriate;
- accuracy: suggestions for measuring the accuracy of the solution, or the stability of the method.

An area where this will work well is with sparse matrix computations. Many important practical problems give rise to large sparse systems of linear equations. One reason for the great interest in sparse linear equations solvers and iterative methods is the importance of being able to obtain numerical solutions to systems of partial differential equations. Such systems appear in studies of electrical networks, economic-system models, and physical processes, such as diffusion, radiation, and elasticity. Iterative methods work by continually refining an initial approximate solution so that it approaches closer and closer to the correct solution. With an iterative method a sequence of approximate solutions $\{x^{(k)}\}$ is constructed that essentially involve the matrix A only in the context of matrix-vector multiplication. Thus, sparsity can be taken advantage of so that each iteration requires $O(n)$ operations.

Many basic methods exist for iteratively solving linear systems and finding eigenvalues. The trick is finding the most effective method for the problem at hand. A method that works well for one problem type may not work as well for another. Or it may not work at all. We have written a book on templates for large sparse linear systems (Barrett et al. 1994) to help address the needs of users of high-performance computers.

5. Software for Large-Scale Eigenanalysis

We have been developing mathematical software for large-scale eigenvalue problems based upon a new variant of the Arnoldi process. Since this is a new algorithm we go into more algorithmic detail than in the sections on dense linear algebra, where the basic algorithms are well known. This new variant of the Arnoldi process employs an implicit restarting scheme that may be viewed as a truncation of the standard implicitly shifted QR-iteration for dense problems. Numerical difficulties and storage problems normally associated with Arnoldi and Lanczos processes are avoided. The algorithm is capable of computing a few eigenvalues with user-specified features, such as largest real part or largest magnitude, using a predetermined storage requirement that is proportional to the matrix order times the desired number of eigenvalues.

The ARPACK software, which is based upon an implementation of this algorithm, has been designed to be efficient on a variety of high-performance computers. Parallelism within the scheme is obtained primarily through the matrix and vector operations that comprise the majority of the work in the algorithm. The software is capable of solving large-scale symmetric, nonsymmetric, and generalized eigenproblems from significant application areas.

5.1 LARGE SPARSE EIGENVALUE SOFTWARE

The most general problem addressed by this software is the generalized eigenproblem

$$Ax = \lambda Mx, \qquad (6)$$

where both A and M are real $n \times n$ matrices and M is symmetric. We assume that the pair (A, M) is a regular definite pencil if A is also symmetric or that M is positive semi-definite if A is nonsymmetric.

Arnoldi's method is a Krylov subspace projection method. It obtains approximations to eigenvalues and corresponding eigenvectors of a large matrix A by constructing the orthogonal projection of this matrix onto the Krylov subspace $Span\{v, Av, \ldots, A^{k-1}v\}$. The Arnoldi process begins with the specification of a starting vector v and in k steps produces the decomposition of an $n \times n$ matrix A into the form

$$AV = VH + fe_k^T, \qquad (7)$$

where v is the first column of the matrix $V \in \mathbf{R}^{n\times k}$, $V^TV = I_k$; $H \in \mathbf{R}^{k\times k}$ is upper Hessenberg, $f \in \mathbf{R}^n$ with $0 = V^T f$, and $e_k \in \mathbf{R}^k$, the kth coordinate basis vector. The vector f is called the residual. This factorization may be advanced one step at the cost of a (sparse) matrix vector product involving A and two dense matrix vector products involving V^T and V. The dense products may be accomplished using Level 2 BLAS. The new column of V will be $v_{k+1} = f/\beta$ where $\beta = \|f\|$, and β will become the next subdiagonal element of H.

The columns of V form an orthonormal basis for the Krylov subspace and H is the orthogonal projection of A onto this space. The eigenvalues and corresponding eigenvectors of H provide approximate eigenvalues and eigenvectors for A. If

$$Hy = y\theta,$$

and we put $x = Vy$, then x, θ is an approximate eigenpair for A with

$$\|Ax - x\theta\| = \|f\| |e_k^T y|,$$

and this provides a means for estimating the quality of the approximation. The information obtained through this process is completely determined by the choice of the starting vector. Eigen-information of interest may not appear until k gets very large. In this case it becomes intractable to maintain numerical orthogonality of the basis vectors V, and it will also require extensive storage. Failure to maintain orthogonality leads to a number of numerical difficulties. Our method provides a means of extracting interesting information from very large Krylov subspaces while avoiding the storage and numerical difficulties associated with the standard approach. It does this by continually compressing the interesting information into a fixed-size k-dimensional subspace. This is accomplished through the implicitly shifted QR mechanism. An Arnoldi factorization of length $k + p$ is compressed to a factorization of length k by applying p implicit shifts, resulting in

$$AV_{k+p}^{+} = V_{k+p}^{+}H_{k+p}^{+} + f_{k+p}e_{k+p}^{T}Q, \qquad (8)$$

where $V_{k+p}^{+} = V_{k+p}Q$, $H_{k+p}^{+} = Q^T H_{k+p}Q$, and $Q = Q_1Q_2 \cdots Q_p$, with Q_j the orthogonal matrix associated with the shift μ_j. It may be shown that the first $k - 1$ entries of the vector $e_{k+p}^T Q$ are zero. Equating the first k columns on both sides yields an updated k-step Arnoldi factorization

$$AV_k^+ = V_k^+ H_k^+ + f_k^+ e_k^T, \qquad (9)$$

with an updated residual of the form $f_k^+ = V_{k+p}^+ e_{k+1} \hat{\beta}_k + f_{k+p}\sigma$. Using this as a starting point it is possible to use p additional steps of the Arnoldi process to return to the original form. Each of these applications implicitly applies a polynomial in A of degree p to the starting vector. The roots of this polynomial are the shifts used in the QR process and these may be selected to filter unwanted information from the starting vector and, hence, from the Arnoldi factorization. Full details may be found in an article by Sorensen (1992).

The resulting software ARPACK based upon this mechanism provides several features that are not present in existing (single vector) codes to our knowledge:

- reverse communication interface;
- ability to return k eigenvalues that satisfy a user-specified criterion, such as largest real part, largest absolute value, largest algebraic value (symmetric case), etc.;
- A fixed pre-determined storage requirement suffices throughout the computation. Usually this is $n * O(2k) + O(k^2)$, where k is the number of eigenvalues to be computed and n is the order of the matrix. No auxiliary storage or interaction with such devices is required during the course of the computation;
- Eigenvectors may be computed on request. The Arnoldi basis of dimension k is always computed. The Arnoldi basis consists of vectors that are numerically orthogonal to working accuracy.
- Accuracy: The numerical accuracy of the computed eigenvalues and vectors is user specified and may be set to the level of working precision. At working precision, the accuracy of the computed eigenvalues and vectors is consistent with the accuracy expected of a dense method such as the implicitly shifted QR iteration.
- Multiple eigenvalues offer no theoretical or computational difficulty other than the additional matrix vector products that are required to expose the multiple instances. This cost is commensurate with the cost of a block version of appropriate blocksize.

"Parallelism within the scheme is obtained primarily through the matrix and vector operations that comprise the majority of the work in the algorithm. The software is capable of solving large-scale symmetric, nonsymmetric, and generalized eigenproblems from significant application areas."

"*One of the most important classes of applications arises in computational fluid dynamics. Here the matrices are obtained through discretization of the Navier-Stokes equations. A typical application involves linear stability analysis of steady-state solutions.*"

5.2 APPLICATIONS OF ARPACK

ARPACK has been used in a variety of challenging applications, and has proven to be useful both in symmetric and nonsymmetric problems. It is of particular interest when there is no opportunity to factor the matrix and employ a "shift and invert" form of spectral transformation,

$$\hat{A} \leftarrow (A - \sigma I)^{-1}. \tag{10}$$

Existing codes often rely upon this transformation to enhance convergence. Extreme eigenvalues $\{\mu\}$ of the matrix $\hat{A}$ are found very rapidly with the Arnoldi-Lanczos process and the corresponding eigenvalues $\{\lambda\}$ of the original matrix A are recovered from the relation $\lambda = 1/\mu + \sigma$. Implementation of this transformation generally requires a matrix factorization. In many important applications this is not possible due to storage requirements and computational costs. The implicit restarting technique used in ARPACK is often successful without this spectral transformation.

One of the most important classes of applications arises in computational fluid dynamics. Here the matrices are obtained through discretization of the Navier-Stokes equations. A typical application involves linear stability analysis of steady-state solutions. Here one linearizes the nonlinear equation about a steady state and studies the stability of this state through examination of the spectrum. Usually this amounts to determining if the eigenvalues of the discrete operator lie in the left half-plane. Typically, these are parametrically dependent problems and the analysis consists in determining phenomena such as simple bifurcation, Hopf bifurcation (an imaginary complex pair of eigenvalues cross the imaginary axis), turbulence, and vortex shedding as this parameter is varied. Our method is well suited to this setting as it is able to track a specified set of eigenvalues while they vary as functions of the parameter. Our software has been used to find the leading eigenvalues in a Couette-Taylor wavy vortex instability problem involving matrices of order 4,000. One interesting facet of this application is that the matrices are not available explicitly and are logically dense. The particular discretization provides efficient matrix vector products through Fourier transform. Details may be found in the article by Edwards et al. (1994).

Alvarez-Cohen and McCarty (1991) studied a groundwater remediation problem through a large nonsymmetric eigenanalysis using a pore-scale model to understand macroscopic groundwater transport phenomena. Convection, diffusion, and biochemical reactions occur at the pore level. The equations model flow through a single pore whose lining reacts with the flowing solute. Boundary conditions are periodic. The eigenvalues of this boundary-value problem provide useful information about the flow through an aggregate of pore cells. The solution of the eigenproblem is discussed by Dykaar (1993). Preliminary computational studies indicate that ARPACK can provide a means of extracting a number of interesting eigenvalues and eigenvectors more efficiently than the inverse power method that is currently employed.

Our software has been used to study the stability of the core of a civil nuclear power plant, as modeled by the two-group neutron diffusion equation. Vaudescal (personal communication 1993) reports improved performance using ARPACK over results obtained in Jaffre and Vaudescal (1993) using explicitly restarted Arnoldi.

Very large symmetric generalized eigenproblems arise in structural analysis. One example that we have worked with at Cray Research through the courtesy of the Ford Motor Company involves an automobile engine model constructed from 3D solid elements. Here the interest is in a set of modes to allow solution of a forced frequency-response problem $(K - \lambda M)x = f(t)$, where $f(t)$ is a cyclic forcing function that is used to simulate expanding gas loads in the engine cylinder as well as bearing loads from the piston connecting rods. The model has over 250,000 degrees of freedom. The smallest eigenvalues are of interest and the ARPACK code appears to be very competitive with the best commercially available codes on problems of this size (for details, see Sorensen, Tomasic, and Vu, 1993).

Nonlinear eigenvalue problems also arise in structural analysis. We are collaborating with researchers at Stanford University in this area. In Smith, Sorensen, and Singh (1993) we present an implicitly restarted Lanczos-based eigensolution technique for evaluating the natural frequencies and modes from frequency-dependent eigenproblems in structural dynamics. The new solution technique is used in conjunction with a mixed finite-element modeling procedure that utilizes both the polynomial- and frequency-dependent displacement fields in formulating the system matrices. The method is well suited to the solution of large-scale problems. The solution methodology presented in Smith et al. (1993) is based upon the ability to evaluate a specific set of parameterized nonlinear eigenvalue curves at given values of the parameter using the symmetric generalized eigensolvers available in ARPACK. Numerical examples illustrate that the implicitly restarted Lanczos method with secant interpolation accurately evaluates the exact natural frequencies and modes of the nonlinear eigenproblem. The examples also verify that the new eigensolution technique, coupled with the mixed finite-element modeling procedure, is more accurate than conventional finite-element models. In addition, the eigenvalue technique presented here is shown to be far more computationally efficient on large-scale problems than the determinant-search techniques traditionally employed for solving exact vibration problems. These techniques are being extended to solve damped problems (which are nonsymmetric) and interior eigenvalue problems.

Computational chemistry provides a rich source of problems. ARPACK is being used in two applications currently and holds promise for a variety of challenging problems in this area. We are collaborating with researchers at Ohio State on large-scale, three-dimensional reactive scattering problems. The governing equation is the Schrödinger equation and the computational technique for studying the physical phenomena relies upon repeated eigenanalysis of a Hamiltonian operator consisting of a Laplacian operator discretized in spherical coordinates plus a surface potential. The discrete operator has a tensor product structure from the discrete Laplacian plus a diagonal matrix from the potential. The resulting matrix has a block structure consisting of $m \times m$ blocks of order n. The diagonal blocks are dense and the off-diagonal blocks are scalar multiples of the identity matrix of order n. It is virtually impossible to factor this matrix directly because the factors are dense in any ordering. We are using a distributed-memory parallel version of ARPACK together with some preconditioning ideas to solve these problems on distributed-memory machines. Encouraging computational results have been obtained on CRAY Y-MP machines and also on the Intel Delta. The code is currently being ported to the CM-5 (see Hayes et al., 1993; Sorensen et al., 1993, for further details).

Nonsymmetric problems also arise in quantum chemistry. Researchers at the University of Washington have used the code to investigate the effects of the electric field on InAs/GaSb and GaAs/Al_xGa_{1-x} as quantum wells. ARPACK was used to find highly accurate solutions to these nonsymmetric problems, which researchers had been unable to solve by other means (see Li and Kuhn, 1993, for details).

Another source of problems arises in magnetohydrodynamics (MHD) in the study of the interaction of a plasma and a magnetic field. The MHD equations describe the macroscopic behavior of the plasma in the magnetic field. These equations form a system of coupled nonlinear PDEs. Linear stability analysis of the linearized MHD equations leads to a complex eigenvalue problem. Researchers at the Institute for Plasma Physics and Utrecht University in the Netherlands have modified the codes in ARPACK to work in complex arithmetic and are using the resulting code to obtain very accurate approximations to the eigenvalues lying on the Alfvén curve. The code is not only finding extremely accurate solutions, it is doing so far more efficiently than the existing method of choice. Currently problems of order 3216 are being solved. The complex version of ARPACK produced 45 good approximations of eigenvalues in 27 seconds of CRAY Y-MP cpu time while the method currently in use needed 32 seconds to find 25 poorly converged approximations (see Kooper et al., 1993, for details).

There are many other applications. In addition to the examples just mentioned, ARPACK has been used to solve large-scale problems in the optimal design of a membrane and in the design of dielectric waveguides. It may also be used to compute the singular value decomposition (SVD) of large matrices. There are many important applications of the SVD, including analysis and enhancement of digital images. Several applications of this technology arise in Computational Biology as well as many other fields. As we gain experience with the ARPACK software, we find an increasing number of new interesting and challenging applications. The dramatic increase in modern computing power, combined with the new algorithms available in the ARPACK software, can provide solutions to eigenproblems that were previously intractable.

6. Conclusions

Linear Algebra is an important part of the research of the CRPC. It impacts almost every part of the effort. The focus of our work is on issues impacting the design of scalable libraries for performing dense and sparse linear algebra computations on multicomputers. The activities provide critical underpinning for much of the work on higher-level optimization algorithms and numerical solution of partial differential equations. The research has proved to be a rich source of basic problems for work on compiler management of memory hierarchies and compiling for distributed-memory machines. Parallelizing compilers should ultimately be able to restructure loops in sequential codes to reproduce the loops of our hand-optimized parallelized codes. Finally, the work has served as a testbed for our ideas on how to design, build, and distribute libraries of scalable mathematical software.

One important factor that has hindered our development of software for distributed memory concurrent computers has been the lack of a widely accepted message passing standard. This led to our initiation of, and involvement in, an effort in the parallel computing community to develop such a standard, called the Message Passing Interface (MPI). The MPI standardization effort involves about 60 people. Most of the major vendors of concurrent computers are involved in MPI, along with researchers from universities, government laboratories, and industry. The CRPC sponsored the first workshop leading to the development of the MPI draft standard in April 1992 (Walker, 1992), and a preliminary draft proposal was put forward by Dongarra, Hempel, Hey, and Walker to foster discussion (Dongarra et al., 1993). A standard message passing interface is a key component in building a concurrent computing environment in which applications, software libraries, and tools can be transparently ported between different machines. MPI provides a number of features that are useful in the design of parallel software libraries. These include support for process groups, application topologies, communication contexts, and general datatypes for messages. For details the reader is referred to the draft MPI standard document (MPI Forum, 1993a) and related papers (MPI Forum, 1993b; Walker, 1994). We intend to develop MPI versions of the BLACS in the near future.

Appendix: Availability of Software

A large body of numerical software is freely available 24 hours a day via an electronic service called *netlib*. In addition to the software discussed here, there are dozens of other libraries, technical reports on various parallel computers and software, test data, facilities to automatically translate Fortran programs to C, bibliographies, names and addresses of scientists and mathematicians, and so on. One can communicate with netlib in one of two ways, by email or (much more easily) via an X-window interface called *xnetlib*. Using email, one sends messages of the form, send subroutine name from library_name or send index for library_name to the address netlib@ornl.gov or netlib@research.att.com. The message will be automatically read and the corresponding subroutine mailed back. xnetlib (which can be obtained and installed by sending the message send xnetlib.shar from xnetlib to netlib@ornl.gov) is an X-window interface that lets one point at and click on subroutines, which are then automatically transferred back into the user's directory. There are also index search features to help find the appropriate subroutine.

To get started using netlib, send the one-line message send index to netlib@ornl.gov. A description of the overall library should be sent to you within minutes (providing all the intervening networks as well as netlib server are up).

Interested parties may obtain the software discussed in this paper by sending email to netlib@ornl.gov and in the email message typing send index from scalapack. Experience with applications is very important to the authors and we welcome the opportunity to work with researchers who want to use the codes.

"The examples verify that the new eigensolution technique, coupled with the mixed finite-element modeling procedure, is more accurate than conventional finite-element models. In addition, the eigenvalue technique presented here is shown to be far more computationally efficient on large-scale problems than the determinant-search techniques traditionally employed for solving exact vibration problems."

ACKNOWLEDGMENT

This work was supported in part by the National Science Foundation Science and Technology Center Cooperative Agreement No. CCR-8809615, by Advanced Research Projects Agency under contract number DAAL03-91-C-0047 administered by the U.S. Army Research Office, and by the Department of Energy under contract number DE-AC05-84OR21400.

BIOGRAPHIES

Jaeyoung Choi is a Postdoctoral Research Associate in the Mathematical Sciences Section of Oak Ridge National Laboratory (ORNL). Dr. Choi received his B.S. from Seoul National University in 1984, his M.S. from the University of Southern California in 1986, and his Ph.D. in Electrical Engineering from Cornell University in 1992. Since March of 1992 Dr. Choi has been working at ORNL, where he is one of the principal designers of the ScaLAPACK software library.

Jack Dongarra holds a joint appointment as Distinguished Professor of Computer Science in the Computer Science Department at the University of Tennessee (UT) and as Distinguished Scientist in the Mathematical Sciences Section at Oak Ridge National Laboratory under the UT/ORNL Science Alliance Program. He specializes in numerical algorithms in linear algebra, parallel computing, use of advanced computer architectures, programming methodology, and tools for parallel computers. Other current research involves the development, testing, and documentation of high-quality mathematical software. He was involved in the design and implementation of the software packages EISPACK, LINPACK, the BLAS, LAPACK, Netlib/XNetlib, PVM/HeNCE, and MPI; and is currently involved in the design of algorithms and techniques for high-performance computer architectures.

Other experience includes work as a senior scientist at Argonne National Laboratory until 1989, a visiting professor at Ecole Normale Supérieure de Lyon in France the summer of 1993, a visiting professor at ETH, Zurich, Switzerland the summer of 1992, a visiting scientist at AERE Harwell Laboratory, England the summer of 1987, a visiting scientist at the Center for Supercomputer Research and Development, University of Illinois, Urbana/Champaign, 1985–1987, a visiting scientist at IBM's T. J. Watson Research Center in 1981, consultant to Los Alamos Scientific Laboratory in 1978, research assistant with the University of New Mexico in 1978, as well as a visiting scientist at Los Alamos Scientific Laboratory in 1977.

Dongarra received a Ph.D. in Applied Mathematics from the University of New Mexico in 1980, a M.S. in Computer Science from the Illinois Institute of Technology in 1973, and a B.S. in Mathematics from Chicago State University in 1972.

Professional activities include membership in the Society for Industrial and Applied Mathematics (SIAM), the Institute of Electrical and Electronics Engineers (IEEE), and the Association for Computing Machinery (ACM). Dongarra has served on the SIAM Council and the ACM SIGNUM Board of Directors. He is also Editor-in-Chief of the *International Journal of Supercomputer Applications and High Performance Computing*, and an editor of *Parallel Computing, Journal of Distributed and Parallel Computing, Journal of Supercomputing, Journal of Numerical Linear Algebra with Applications*, and *Communications of the ACM*.

He has published numerous articles, papers, reports and technical memoranda, and has given many presentations on his research interests.

Roldan Pozo is a Research Associate and Adjunct Professor in the Department of Computer Science at the University of Tennessee, Knoxville. He researches issues in the design of software tools for scientific computing on parallel architectures. He is the principal designer of LAPACK++, an object-oriented software library for solving linear systems on workstations and parallel architectures. Dr. Pozo has been involved with parallel software development at the Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique (CERFACS) in Toulouse, France, and at the National Center for Atmospheric Research in Boulder, Colorado. He received his Ph.D. in Computer Science from the University of Colorado in 1991.

Danny C. Sorensen received his B.S. in Mathematics from the University of California, Davis in 1972 and his Ph.D. in Mathematics from the University of California, San Diego in 1977. He was an Assistant Professor of Mathematics at University of Kentucky from 1977–1980. He then joined the Mathematics and Computer Science Division of Argonne National Laboratory, where he became a Senior Computer Scientist. He joined the faculty of Rice University in 1989 where he is Professor of Mathematical Sciences.

David W. Walker is a Research Staff Member in the Mathematical Sciences Section at Oak Ridge National Laboratory (ORNL), and an Adjunct

Associate Professor in the Department of Computer Science of the University of Tennessee, Knoxville. He obtained his B.A. in Mathematics from Jesus College, Cambridge, in 1973, his M.S. in Astrophysics in 1979, and his Ph.D. in Physics from the University of London in 1983. From 1986–8 Dr. Walker was a member of the research staff of the Concurrent Computation Project at the California Institute of Technology, and from 1988–1990 was an Associate Professor in the Department of Mathematics at the University of South Carolina. He has worked at ORNL since 1990, where he is mainly involved in the design of software libraries, algorithms, and applications for distributed-memory concurrent computers.

SUBJECT AREA EDITOR

Monica Lam

REFERENCES

Alvarez-Cohen, L. M. and McCarty, P. L. 1991. A cometabolic biotransformation model for halogenated aliphatic compounds exhibiting product toxicity. *Environmental Science and Technology* 25(8):1381–1387.

Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Croz, J. D., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S., and Sorensen, D. 1992. *LAPACK User's Guide.* Philadelphia: SIAM.

Barrett, R., Berry, M., Chan, T. F., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., and van der Vorst, H. 1994. *Templates for the solution of linear systems.* Philadelphia: SIAM.

Choi, J., Dongarra, J. J., Pozo, R., and Walker, D. W. 1992. ScaLAPACK: A scalable linear algebra library for distributed memory concurrent computers. In *Proc. fourth symposium on massively parallel computing*, edited by H. J. Siegel, pp. 120–127.

Choi, J., Dongarra, J. J., and Walker, D. W. 1993a. The design of scalable software libraries for distributed memory concurrent computers. In *Environments and tools for parallel scientific computing*, edited by J. J. Dongarra and B. Tourancheau. Proc. workshop held September 7–8, 1992, in Saint Hilaire du Touvet, France, pp. 3–15.

Choi, J., Dongarra, J. J., and Walker, D. W. 1993b. Level 3 BLAS for distributed memory concurrent computers. In *Environments and tools for parallel scientific computing*, edited by J. J. Dongarra and B. Tourancheau. Proc. workshop held September 7–8, 1992, in Saint Hilaire du Touvet, France, pp. 17–29.

Choi, J., Dongarra, J. J., and Walker, D. W. 1994a. The design of a parallel, dense linear algebra software library: Reduction to Hessenberg, tridiagonal, and bidiagonal form. In preparation.

Choi, J., Dongarra, J. J., and Walker, D. W. 1994b. PUMMA: Parallel universal matrix multiplication algorithms on distributed memory concurrent computers. *Concurrency: Practice and experience.* To appear. (Also available as Oak Ridge National Laboratory report TM-12252, November, 1992.)

Dongarra, J., Du Croz, J., Duff, I., and Hammarling, S. 1990. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Software* 16: 1–17.

Dongarra, J., and Ostrouchov, S. 1990. LAPACK block factorization algorithms on the Intel iPSC/860. Technical Report CS-90-115, University of Tennessee at Knoxville, Computer Science Department.

Dongarra, J. J. 1991. LAPACK Working Note 34: Workshop on the BLACS. Computer Science Dept. Technical Report CS-91-134, University of Tennessee, Knoxville, TN. (LAPACK Working Note No. 34).

Dongarra, J. J., Du Croz, J., Hammarling, S., and Hanson, R. 1988. An extended set of Fortran basic linear algebra subroutines. *ACM Trans. Math. Software* 14(1): 1–17.

Dongarra, J. J., Hempel, R., Hey, A. J. G., and Walker, D. W. 1993. A proposal for a user-level, message passing interface in a distributed memory environment. Technical Report TM-12231, Oak Ridge National Laboratory.

Dongarra, J. J., Pozo, R., and Walker, D. W. 1993. Design overview of object-oriented extensions for high performance linear algebra. In *Proc. Supercomputing '93*, IEEE Computer Society Press.

Dongarra, J. J., van de Geijn, R., and Walker, D. W. 1992. A look at scalable dense linear algebra libraries. In *Proc. scalable high-performance computing conference*, pp. 372–379. IEEE Publishers.

Dongarra, J. J., and van de Geijn, R. A. 1991. Two-dimensional basic linear algebra communication subprograms. Technical Report LAPACK Working Note 37, Computer Science Department, University of Tennessee, Knoxville, TN.

Dongarra, J. J., van de Geijn, R. A., and Walker, D. W. 1994. Scalability issues affecting the design of dense linear algebra library. *Journal of Parallel and Distributed Computing.* Accepted for publication.

Dykaar, B. 1993. Macroscopic groundwater flow and transport coefficients. Ph.D. Thesis Proposal, Stanford University.

Edelman, A. 1993. Large dense numerical linear algebra in 1993: The parallel computing influence. *International Journal Supercomputer Applications.* 7(2):113–128.

Edwards, W. S., Tuckerman, L. S., Friesner, R. A., and Sorensen, D. C. 1993. Krylov methods for the incompressible Navier-Stokes equations. *Journal of Computational Physics* 110(1):82–102.

Fox, G. C., Johnson, M. A., Lyzenga, G. A., Otto, S. W., Salmon, J. K., and Walker, D. W. 1988. *Solving problems on concurrent processors*, Vol. 1. Englewood Cliffs, N.J.: Prentice Hall.

Harrington, R. 1990. Origin and development of the method of moments for field computation. *IEEE Antennas and Propagation Magazine.*

Hayes, E. F., Pendergast, P. H., Darakjian, Z., and Sorensen, D. C. 1993. Scalable algorithms for three-dimensional reactive scattering: evaluation of a new algorithm for obtaining surface functions. *J. of Computational Physics.* To appear.

Hess, J. L. 1990. Panel methods in computational fluid dynamics. *Annual Reviews of Fluid Mechanics* 22:255–274.

Hess, J. L., and Smith, M. O. 1967. Calculation of potential flows about arbitrary bodies. In *Progress in Aeronautical Sciences*, Vol. 8, edited by D. Küchemann, Pergamon Press.

Jaffre, J., and Vaudescal, J.-L. 1993. Arnoldi's method for two-group neutron diffusion. In *Proc. international conference on mathematical methods and supercomputing in nuclear applications*, pp. 19–23.

Kooper, M. N., van der Vorst, H. A., Poedts, S., and Goedbloed, J. P. 1993. Application of the implicitly updated Arnoldi method with a complex shift and invert strategy in mhd. Technical report, Institute for Plasmaphysics, FOM Rijnhuizen, Nieuwegein, The Netherlands.

Lawson, C., Hanson, R., Kincaid, D., and Krogh, F. 1979. Basic linear algebra subprograms for Fortran usage. *ACM Trans. Math. Software* 5:308–323.

Li, T. L., and Kuhn, K. J. 1993. Finite element solution to quantum wells by irreducible formulations. Technical report, University of Washington, Seattle.

Lichtenstein, W., and Johnsson, S. L. 1993. Block cyclic dense linear algebra. *SIAM Journal on Scientific Computing* 14(6):1259–1288.

MPI Forum, T. 1993a. Document for a standard message-passing interface. Technical Report CS-93-214, Department of Computer Science, University of Tennessee, Knoxville. Also available electronically using netlib.

MPI Forum, T. 1993b. MPI: A message passing interface. In *Proc. Supercomputing 93*. IEEE Computer Society Press.

Smith, H. A., Sorensen, D. C., and Singh, R. K. 1993. A Lanczos-based eigensolution technique for exact vibration analysis. *International Journal for Numerical Methods in Engineering* 36:1987–2000.

Sorensen, D. C. 1992. Implicit application of polynomial filters in a *k*-step Arnoldi method. *SIAM Journal on Numerical Analysis (Series B)* 28:1752–1775.

Sorensen, D. C., Tomasic, Z. A., and Vu, P. A. 1993. Algorithms and software for large scale eigenproblems on high performance computers. In *Proc. high performance computing 93*, edited by A. Tentner, Society for Computer Simulation, pp. 149–154.

Van de Velde, E. F. 1990. Data redistribution and concurrency. *Parallel Computing* 16.

Walker, D. 1992. Standards for message passing in a distributed memory environment. Technical Report TM-12147, Oak Ridge National Laboratory.

Walker, D. W. 1994. The design of a standard message passing interface for distributed memory concurrent computers. *Parallel Computing* 20(4):657–673.

Wang, J. J. H. 1991. *Generalized moment methods in electromagnetics.* New York: John Wiley & Sons.