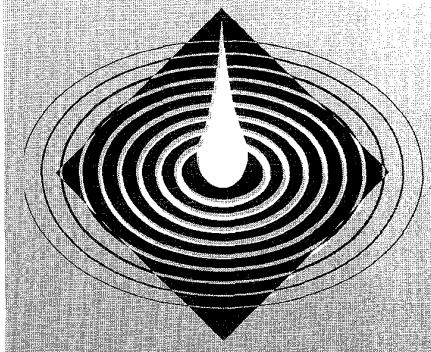

Essays by
Jack Dongarra
Eric Grosse
Yale N. Patt
K. Mani Chandy
Yoichi Muraoka

Taking Stock, Looking Ahead

Future Linear-Algebra Libraries

Jack Dongarra
University of Tennessee, Knoxville

The ultimate development of fully mature, parallel scalable libraries will necessarily depend on breakthroughs in many supporting technologies. Scalable library development cannot wait, however, until all the enabling technologies are in place for two reasons: The need for such libraries for existing and near-term parallel architectures is immediate, and progress in all the supporting technologies depends on feedback from concurrent efforts in library development.

Traditional libraries

We in the linear-algebra community have long recognized that we needed something to help us develop our algorithms into software libraries. Several years ago, as a community effort we put together a de facto standard for identifying the basic operations required in our algorithms and software. Our hope was that many manufacturers would implement the standard on their machines, and that we could then draw on the power of that implementation in a rather portable way. We began with BLAS operations (basic linear-algebra subprograms) designed for basic matrix computation. Since message passing is critical in a parallel system, we worked on developing message-passing standards. Both the PVM (parallel virtual machine) and the MPI (message-passing interface) have helped establish standards and promote portable software, critical for software library work.

User interfaces

As computer architectures and programming paradigms become increasingly complex, we want to hide this complexity from users as

1070-9924/96/$5.00 © 1996 IEEE

much as possible. The traditional user interface for large, general-purpose mathematical and scientific libraries is a user-written program (usually in Fortran or C) that calls on library routines to solve specific subproblems that arise during the computation. When extended to run on parallel architectures, this approach has only a limited ability to hide the underlying architectural and programming complexity from users. As we extend the conventional notion of mathematical and scientific libraries to scalable architectures, we must rethink the conventional user-interface concept and devise alternate approaches that hide architectural, algorithmic, and data complexity from users.

One promising approach is a problem-solving environment, typified by current packages such as Matlab, that provides an interactive, graphical interface for specifying and solving scientific problems. Algorithms and data structures are hidden from the user because the package itself efficiently and distributively stores and retrieves the problem data. This approach seems especially appropriate in view of the trend toward graphics workstations as the primary user access to computing facilities. Ultimately, such an interface can provide seamless access to networks including various parallel computers and conventional supercomputers. These computational engines would be invoked selectively for different parts of the user's computation, depending on which machine is most appropriate for a particular subproblem. I envision at least two interfaces for a linear-algebra library: one would follow conventional lines (Lapack-style) for immediate use in conventional programs ported to novel machines, and the other would take the form of a problem-solving environment (Matlab-style). These proposed interfaces are not incompatible; in fact, the problem-solving environment can be built on top of software based on a more conventional interface.

Heterogeneous networking

Current trends in parallel architectures, high-speed networks, and personal workstations suggest that the future computational environment of working scientists will require the seamless integration of heterogeneous systems into a coherent problem-solving environment. Graphical workstations will provide the standard user interface, with a variety of computational engines and data storage devices distributed across a network. Because of the diversity of parallel architectures, different computational tasks will be more efficient on some than on others, with no single architecture uniformly superior. Thus, I expect the problem-solving environment eventually to migrate to a heterogeneous network of workstations, file servers, and parallel computation servers. The various computational tasks required to solve a problem will automatically and transparently be targeted to the most appropriate computational engine on the network. Many users will share system resources, but in a manner somewhat different from conventional time-sharing.

We have already made important first steps toward achieving these goals with systems such as PVM and MPI, which supply the low-level services necessary to coordinate the use of multiple workstations and other computers for individual jobs. Either system could serve as the foundation for a complete problem-solving environment. Network computing services such as NetSolve look for computational resources on a network for a submitted problem (which can be a single Lapack, Scalapack, or Matlab function call), choose the best resource available, solve the problem (with retry for fault tolerance), and return the answer to the user. NetSolve is available for Fortran, C, and Matlab users.

Software tools and standards

An ambitious development effort in scalable libraries will require a great deal of supporting infrastructure. Moreover, any library's portability depends critically on adherence to standards. In the case of software for parallel architectures, precious few standards exist, so new standards must evolve along with research and development. A particularly important area for scalable distributed-memory architectures is internode communication. The BLAS standards have proven very effective in assisting the development of portable, efficient software for sequential computers and some of the current class of high-performance computers. We are investigating the possibility of expanding this set of standards, and we also need a lightweight interface to many traditional BLAS functions. In addition, iterative and sparse direct methods require additional functions not provided in traditional BLAS. Numerical methods for dense matrices on parallel computers require high-efficiency kernels that provide functionality similar to that of traditional BLAS on sequential

machines. Software tools are also important, both to help developers design and tune library software and to help users monitor the efficiency of their applications.

Despite the lack of enabling technologies, library development cannot wait for research in programming languages, compilers, software tools, and other areas to mature; it must proceed in conjunction with this research. The time to begin is now.

I see four major research issues in developing parallel scalable linear-algebra libraries. First, the user–library interface needs rethinking; the conventional library interface is inadequate for hiding underlying complexity from users. Second, object-oriented programming will be necessary to develop portable libraries that allow users to work at an appropriate conceptual level. Next, work on algorithms, particularly in linear algebra, is important and cannot be isolated from general library development. Finally, the lack of language standards is the most significant obstacle to the development of communication libraries. A language standard must emerge before a software tool "development sweep" can begin. ♦

Jack Dongarra is a distinguished professor at the University of Tennessee and an adjunct professor at Rice University, both in computer science, and a distinguished scientist in the Mathematical Sciences Section at Oak Ridge National Lab. He participated in the design and implementation of Eispack, Linpack, BLAS, Lapack, Scalapack, Netlib/XNetlib, PVM/Hence, MPI, and the National High-Performance Software Exchange. He can be reached at Univ. of Tennessee, 104 Ayres Hall, Knoxville, TN 37996; e-mail, dongarra@cs.utk.edu, WWW, http://www.netlib.org/utk/people/JackDongarra.html.

Network Programming and CSE

Eric Grosse
Bell Laboratories

A remarkable feature of scientific computing has been its relatively strong reuse of software, and therefore its concern with portability. Early examples of this are the Eispack, Linpack, and Funpack projects, the PFORT verifier, and the BLAS. Thanks to such efforts and to standardization of languages and floating-point hardware, we can now write core computational modules in C or Fortran with confidence that our algorithms will compile and run properly on all the dominant computers today and for the foreseeable future. Some things could still be improved, such as language support for the IEEE floating-point standard, but by and large the situation is satisfactory. This common environment has made possible the extensive catalogs of reusable components in Netlib, *Numerical Recipes*, NAG, IMSL, and so on.

Achieving a similar portability in the rest of scientific computing remains a challenge. Graphics, interprocess communication, network naming rules, and database interfaces are all in flux. For a while in the late 1980s it appeared that the scientific community was converging on Unix, X, and a few portable communication libraries, but now even this consensus is breaking down. The computing world is changing rapidly and unpredictably. At the moment the most promising development is the birth of systems like Java and Inferno that extend the scope of portable programming to include graphical user interfaces, simple visualization, and network services. (Another welcome benefit of these systems is bounds checking of array references, a common amenity in scientific programming in the early '70s but less well sup-