Technical Editor:
Marcin Paprzycki
Dept. of Computer Science
and Statistics
Univ. of Southern Mississippi
Southern Station 1506
Hattiesburg, MS 39406-1506
m.paprzycki@usm.edu

Using Plapack: A solid part of the *Scientific and Engineering Computation* series

By Roman Wyrzykowski, Technical University of Czestochowa

Using Plapack: Parallel Linear Algebra Package
Robert A. Van de Geijn
225 pages
$30.00
MIT Press
Cambridge, Mass.
1997
ISBN 0262720264

The dominant cost in most large-scale computational science applications comes from writing new code. A pragmatic approach to overcoming this difficulty is to create an application-specific package (or library)—a software system that provides a coding interface to solve a target class of problems without focusing on implementation details. This approach originates from the era of sequential computers but has become especially popular for parallel machines, because their low-level programming was (and still is) a formidable task.

The library-based approach to programming parallel computers is especially widespread in computational linear algebra. Numerous parallel linear algebra packages exist, all differing in their orientation (a shared or distributed memory model of parallel programming), functionality (using direct or iterative methods, solving linear systems or eigenvalue problems), exploitation of sparsity of data, availability of support for using linear algebra methods in solving real-life problems, and so forth. Among these packages are Lapack and ScaLapack, Plapack, Aztec, PSPARSLIB, PETSc, BlockSolve, and Parpack.

Using Plapack: Parallel Linear Algebra Package is part of the MIT Press's renowned *Scientific and Engineering Computation* series. The series includes such well-known publications as *Using MPI: Parallel Programming with the Message-Passing Interface* by William Gropp, Ewing Lusk, and Anthony Skejellum (1994) and *PVM: Parallel Virtual Machine*, by Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam (1994), both of which have played an important role in the dissemination of the message-passing parallel-programming model. Van de Geijn's book follows the pattern established by these successful publications.

How to build and use Plapack

Van de Geijn's goal is to deliver the principles for building and using Plapack. This portable parallel linear-algebra library, developed at the University of Texas in Austin, is based on the MPI. However, thanks to object-based programming, Plapack hides many particulars from the user, such as indexing and the implementation details of data distribution and exchange.

Chapter 1 offers the most significant information, concisely summarizing Plapack's main ideas. It starts with the recursive formulation of the Cholesky factorization, and this simple example shows what programmers need to code parallel linear-algebra algorithms. The next four chapters—of which two and five are the most important—contain a guide to the basic routines of the Plapack infrastructure. Chapter 2 acquaints the reader with the routines Plapack provides. These routines initialize the environment, create a template describing the distribution of vectors and matrices among a logical topology of nodes, and create linear algebra objects that encapsulate the distributed vectors and matrices. Chapter 5 details the copy and reduce operations that Plapack uses to hide communication. Chapters 6 through 8 discuss the implementation of routines contained in the *Basic Linear Algebra Subprograms* library. Their contents are structured in accordance with the

three levels of BLAS: vector–vector, matrix–vector, and matrix–matrix operations, respectively. Each chapter describes both the local and global versions of BLAS calls for Fortran and C binding and concludes with examples that clarify the principles and details of using these calls in practice.

Chapter 9 illustrates the use of the Plapack routines in linear algebra applications by discussing in detail several approaches to the parallelization of the Cholesky factorization, namely level-2 and level-3 BLAS implementations for the right-looking and left-looking variants. The corresponding codes reflect algorithms given in standard textbooks in a natural, unsophisticated manner. However—even in the most advantageous implementation—when using a sequence of Plapack level-3 BLAS routines, considerable inefficiency could occur when increasing the number of messages and temporary work buffers. Van de Geijn shows how to reduce this inefficiency for the left-looking level-3-based algorithm, which is reimplemented at a lower level. However, this makes the resulting implementation considerably more complex.

The book closes with two appendices containing summaries of the basic and BLAS-related routines of Plapack and their calling sequences.

A USEFUL BUT LIMITED GUIDE

Using Plapack offers a well-balanced view of the Plapack parallel software, including its implementation and use. Consequently, it proves very useful to its primary readers—scientific-application programmers. For the same reasons, it has interested developers of parallel libraries in various application areas. Additionally, I recommend it for undergraduate and postgraduate students in different disciplines. The high level of the provided routines and easy access to such a comprehensive guide make Plapack a good tool for parallel scientific computing education.

However, from an educational perspective, I question the extremely condensed presentation of BLAS. Independent of whether this was Van de Geijn's decision or the result of editorial constraints, this shortcoming could be redressed to a certain extent by including in the bibliography references to appropriate textbooks such as *Numerical Linear Algebra for High-Performance Computers* by Jack Dongarra, Lain Duff, Danny Sorensen, and H. Van der Vorst.

Unfortunately, the book lacks an overview of its contents—a serious disadvantage to many readers. This makes the material difficult to read and understand, especially for those who are not familiar with parallel dense linear-algebra libraries. An overview would have let readers choose for themselves the most appropriate path through the book.

Another problem concerns the limited number of examples. Fortunately, Van de Geijn refer readers to the Plapack Web page (www.cs.utexas.edu/users/plapack), where you can access the Plapack public-domain software and documentation. In particular, this page provides information on the project's current status, additional calls, and codes for all the examples presented in each chapter. Other examples include LU factorization with pivoting and Householder QR factorization.

Cluster Computing

Continued from page 11

- is publishable in its own right as a new scientific result independent of the fact that the result was mechanically created;
- is equal to or better than the most recent human-created solution to a long-standing problem for which there has been a succession of increasingly better human-created solutions;
- is equal to or better than a result that was considered an achievement in its field at the time it was first discovered;
- solves a problem of indisputable difficulty in its field; or
- holds its own or wins a regulated competition involving human contestants (in the form of either live human players or human-written computer programs).

PARALLELIZATION

The major component of the computational burden of solving nontrivial problems with the GA or genetic programming is the task of measuring each individual's fitness in each generation of the evolving population. Because the task of measuring the fitness of one particular individual in the population is decoupled from the task of measuring the fitness of all other individuals, we can realize nearly 100% efficiency from a parallel computer system running GAs and genetic programming. In fact, the decoupled nature of GAs and genetic programming usually also extends to the level of fitness cases.

Given the inherent parallelism of genetic programming, various schemes for mapping genetic programming onto multiple processors are possible, including mapping

- a semi-isolated subpopulation to each processor (island model),
- a single individual to each processor,
- a single fitness case to each processor,
- a single time step to particular processors, and
- a particular independent run to a processor.

A mapping can be static or dynamic. Similarly, there are many possible schemes for the processors to communicate and exchange information about the solutions. One approach is to let each processor work totally independently and communicate only at the end to select the best solution among all processors. However, a processor might waste all its processing cycles when it gets stuck at a poor population without the knowledge that other processors are searching in some more promising search-space regions. This might reduce the diversity of the chromosomes in the search space. We could also let the processors communicate periodically to exchange the information about their solutions found thus far and then broadcast the best solution to all the processors. However, the information exchange can be a significant overhead that limits the achievable speedup.

Ishfaq Ahmad is an associate professor in the Computer Science Department at the Hong Kong University of Science and Technology. He received a BSc in electrical engineering from the University of Engineering and Technology, Lahore, Pakistan, and an MS in computer engineering and PhD in computer science from Syracuse University. Contact him at iahmad@cs.ust.hk.