NUMERICAL METHODS IN APPLIED SCIENCES

Edited by

Wei Cai **Zhongci Shi**
Chiwang Shu **Jinchao Xu**

SCIENCE PRESS, NEW YORK, BEIJING

O24
N971

Numerical Methods in Applied Sciences

Edited by

Wei Cai Zhongci Shi
Chiwang Shu Jinchao Xu

SCIENCE PRESS
NEW YORK, BEIJING

Responsible Editor: Lin Peng

Published by
Science Press New York, Ltd.
84-04 58th Ave
Elmhurst, NY 11373
USA

ISBN 1-880132-15-x (New York)
ISBN 7-03-005152-1 (Beijing)

Printed in USA

中国科学院科学出版基金资助出版

Numerical Methods in Applied Sciences

Preface

This monograph is composed of fifteen solicited and peer-reviewed articles for an verview of theoretical and algorithmic developments of selectedactive research areas in numerical analysis and computational sciences.

The authors of these articles are all overseas Chinese researchers.The book is a result of our strong desire to contribute our knowledge to the advancement of scientific and technological development in our motherland. We believe, in this information era, that scientific prominence in China relies to a large extent on researchers having the most current developments in the world accessible to them. By exposing our counterpart colleagues and especially graduate students in China to some of the most updated research materials, we hope that this book will provide one of the many avenues in providing the aforementioned accessibility.

With the ever increasing importance of computers' role in scientific research and industrial applications, a tremendous amount of research has been done in recent years for the development of faster and more efficient numerical algorithms for modern high performance computers. The goal of this book is to identify and give a bird eye view of some of the most active areas in these research activities. Because of the limitation of our time, resources and also the size of the book, there are perhaps additional important subjects in this area that have not been addressed in this book. Most of the papers are written mainly based on the authors' own research experiences but with efforts to point out to the readers the developments in other related research directions. We hope this book will be a useful research and teaching reference for graduate students and researchers in applied mathematics, engineering and any other computationally related disciplines.

We would like to thank all the authors for their contributions to this volume and especially the many anonymous reviewers for their expertise which has improved the quality of this book.

Wei Cai
Zhong-Ci Shi
Chi-Wang Shu
Jinchao Xu

July, 1995

Contributors

Zhaojun Bai
Department of Mathematics, University of Kentucky, Lexington, KY 40506

Gang Bao
Department of Mathematics, University of Florida, 201 Walker Hall, Gainesville, FL 32611

Wei Cai
Department of Mathematics, University of California, Santa Barbara, CA 93106

Qiang Du
Department of Mathematics, Michigan State University, East Lansing, MI 48824

Weinan E
Courant Institute of Mathematical Sciences, New York , 251 Mercer Street, New York, NY 10012

Zhong Ge
The Fields Institute for Research in Mathematical Sciences, 185 Columbia St. West, Waterloo, Ontario N2L 5Z5, Canada

Ming Gu
Department of Mathematics and Lawrence Berkeley Laboratory, University of California, Berkeley, CA 94720

Thomas Yizhao Hou
Applied Mathematics, 217-50, California Institute of Technology, Pasadena, CA 91125

Jian-Guo Liu
Department of Mathematics, Temple University, Philadelphia, PA 19122

Ling Ma
Department of Mathematics, Carnegie-Mellon University, Pittsburgh, PA 15213

Chi-Wang Shu
Division of Applied Mathematics, Brown University, Providence, Rhode Island 02912

Junping Wang
Department of Mathematics, University of Wyoming, Laramie, WY 82071

Jinchao Xu
Department of Mathematics, Penn State University, University Park, PA 16802

Yinyu Ye
Department of Management Sciences, The University of Iowa, Iowa City, IA 52242

Hongyuan Zha
Department of Computer Science and Engineering, Penn State University,
University Park, PA 16802

Zhiming Zhang
Department of Mathematics, Texas Tech University, Lubbock, TX 79409

Table of Contents

Parallel Matrix Computations

ZHAOJUN BAI

ABSTRACT. With the increasing availability and usage of advanced-architecture computers, parallel matrix computations is an exceedingly active research and development field. Matrix computations lie at the heart of many scientific computing problems. New architectures impact all traditional goals of algorithm design and analysis and software development for matrix computation problems. In this paper, we give a snapshot of the recent development of parallel matrix computations. We discuss the main features of today's high performance computer systems which impact the parallel algorithm and software design and development. We describe the recent development of high performance libraries, templates and toolboxes for the most common linear algebra problems. We use divide and conquer type algorithms for eigenvalue problems as examples to show that software development is an excellent way to stimulate research.

1. Introduction

High performance parallel computing capability offers the promise of great advances in the design of high speed civil transport, rational drug design, semiconductor device and process simulation, global climate modeling, and other applications [**38**]. The increasing availability of advanced-architecture computers is having a very significant impact on all spheres of scientific computation. Linear algebra and matrix computation problems lie at the heart of most scientific computing. Reliable, efficient, portable and ease-of-use matrix computation algorithms and software provide an infrastructure to satisfy the needs of computational scientists and engineers. In this paper, we will review some recent development of parallel algorithms and software for the most common problems in numerical linear algebra (matrix computations). Particular emphasis is placed on the development of

1991 *Mathematics Subject Classification.* Primary 65F15, 65F10.

The author was supported in part by NSF grant ASC-9102963 and in part by ARPA grant DM28E04120 via a subcontract from Argonne National Laboratory.

algorithms and software packages which are available in the public domain.

We begin with a brief outline of the main features of today's high performance computer systems, general principles in parallel algorithm design, and tradeoffs in parallel algorithm and software development. Then, we give a brief description of the linear algebra libraries, such as EISPACK, LINPACK and LAPACK, followed by ScaLAPACK, which is currently under development. The motivation and importance of the development of LAPACK and ScaLAPACK for high performance computer systems are stressed. The main design strategies of these high performance linear algebra packages are outlined.

It is quite difficult to accommodate all mathematical software users with the same sequential software, and this problem is exacerbated on parallel machines because of the heightened need for performance tuning and larger variety of data layouts. In section 4, we will discuss the development of the *templates* and *toolboxes* around the sparse matrix computations. These templates and toolboxes can easily be assembled to adapt to a particular machine and particular application.

Underlying all these developments is the design and analysis of numerical algorithms. Although the purpose of a software development project is, aside from the production of the package itself, to investigate the problems involved in the development of high quality mathematical software, a lot of interesting and challenging research problems have emerged. In [**66, 17**], Stewart and Demmel have proposed a number of open problems with respect to the development of LINPACK and LAPACK. In section 5, we will use divide and conquer type algorithms for eigenvalue problems as examples to show that software development is an excellent way to stimulate research.

This review is by no means an exhaustive presentation. The author will focus on those developments with which he is involved and familiar. It is evidently biased by the prejudices of the author and his ignorance of many areas of an ever growing literature. The reader might consult some other recent survey papers, such as [**31**] and [**19**], where a bibliography of over two thousands entries is included.

2. High Performance Computer Systems

Flynn's classification of parallel computational models, the one widely used today, views the conventional serial computation model (von Neumann model) as a Single stream of Instruction controlling a Single stream of Data (SISD). One can take one step toward parallelism by introducing Multiple Data streams (SIMD) and a second step by adding Multiple Instruction steams (MIMD).

Thinking Machine's CM-2 and MasPar's MP-2 are the examples of SIMD ma-

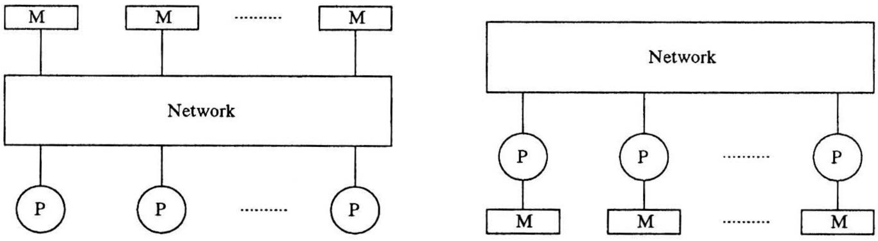

FIGURE 1. MIMD parallel architectures: shared memory (LEFT) and distributed memory (RIGHT)

chine. In general, SIMD machines are particularly suited to special purpose computing.

In the MIMD parallel computational model, as shown in Figure 1, there are two ways to arrange the processing elements (P) and memory modules (M). The configuration with a shared memory model of computation is that every processing element has equal access to all memory. In the distributed memory configuration, the processing elements each have their own memory and communicate by "message-passing". A shared memory MIMD machine hides the interconnection from users and is ease-of-use. However, it is limited by the number of processor elements allowed. Cray Y-MP and C-90, IBM 3090 and 9000, and Convex C-series are shared memory machines. The Intel Paragon and Thinking Machines' CM-5 and the more recently announced IBM SP1 and Cray T3D are distributed memory concurrent supercomputers.

All machines, even desktop PCs and workstations, have memory hierarchies, from fast and small registers, to caches, then to slower and larger main memory. For distributed memory machines, the memory hierarchy also includes the off-processor memory of the other processors. The amount of time it takes to move the data from the memory to the operation unit can far exceed the time required to perform the operation unless the memory is immediately proximate to the operation unit, such as in a register or cache. The actual performance of a vector or a scalar floating point unit is often limited by the rate of transfer of data between different levels of memory in a machine.

The reader is referred to two recent excellent textbooks, [**37**, **45**], on advanced computer architectures.

An emerging computing platform is to use a set of software which allow a heterogeneous collection of computers, such as desktop workstations, hooked together by a network, to be used as a parallel computer. We may give it the nickname "poor man's supercomputer". A program uses message passing routines to communicate and synchronize with other programs. By sending and receiving messages, multiple processes can cooperate to solve a problem in parallel. PVM (an acronym for Parallel Virtual Machine) is one of such software products available in the public domain [**34**]. PVM was initiated at the Oak Ridge National Laboratory, and is now an ongoing research project involving a number of academic institutions and national laboratories and supported by many computer vendors, such as Cray, HP, IBM and Intel.

2.1. General principles in parallel algorithm design. In a sequential computation environment, the execution time of an algorithm is basically proportional to the number of floating point operations it performs. What are the general principles to guide us in design and analysis of parallel algorithms on advanced-architecture concurrent computers? The answers are the *locality* and *regularity* of operation [**19**].

Locality refers to the proximity of the arithmetic and storage components of computers. All machines have a memory hierarchy. Useful arithmetic and logical operation can occur only on data at the top of the memory hierarchy. It is desirable to minimize the time spent moving data between levels of the memory hierarchy, and to use data as much as possible while it is stored in the higher levels of the memory hierarchy.

Regularity means that the operations (including arithmetic and logical operations and communication) which parallel machines perform fastest tend to have simple and regular patterns, such as matrix-matrix multiplication. Efficiency demands that computations be decomposed into repeated applications of these patterns. It is one of the major challenges to design a parallel algorithm which uses a very high fraction of these regular operations, in addition to maintaining locality.

We should note that there is one factor determining the actual program's running time, it is *Amdahl's Law*. As Amdahl noted, the computing time can be divided into the parallel portion and the serial portion. The computation speedup approaches a constant limit determined by the serial portion, no matter how high the degree of the parallelism.

2.2. Different user groups and important tradeoffs. With the rapid advances of computer facilities, in particular, massively parallel computers, and the new engagement of interdisciplinary scientific computing activities, there are two different groups of mathematical algorithm and software users according to their different desired priorities in terms of computation details, reliability and execution time of a program. The first group is made up of traditional library users and the second group is made up of high performance computing researchers. For the first group the desiderata can be characterized as follows:

(i) easy user interface with hidden computational details,
(ii) reliability; the code should fail as rarely as possible,
(iii) execution time.

However, for the second group, the desiderata are:

(i) execution time,
(ii) being able to access internal details to fine tune data structures to one's applications,
(iii) reliability; a program should expend only a negligible amount of time, space or code in checking or taking precautions against rare eventualities that the user knows may never arise for his or her particular applications.

These different desiderata give an extra dimension to numerical algorithm development and analysis. To what extent can we satisfy both groups? In this paper, we will see how to address this interesting question with respect to matrix computations.

To this end, we should note that because of the constraints of locality and regularity of operations and other factors in parallel computation, certain tradeoffs have to be made in algorithm design and implementation. One such tradeoff is *time* versus *space*. An algorithm which uses less space may have to go more slowly. Another interesting tradeoff is *parallelism* versus *numerical stability* [**18**]. For some problems, highly parallel algorithms are known to be as less numerically stable than conventional sequential algorithm. For example, Sameh and Brent's algorithm for solving an $n \times n$ linear system with a triangular matrix T takes $O(\log^2 n)$ parallel steps [**61**]. However, the error analysis of this algorithm shows an error bound proportional to $\kappa^3(T)\epsilon$, where $\kappa(T) = \|T\| \, \|T^{-1}\|$ is the condition number of T (note that $\kappa(T) \geq 1$), and ϵ is machine precision. This is in contrast to the error bound of $\kappa(T)\epsilon$ for the usual sequential substitution algorithm. We will see the same kind of tradeoffs for the different algorithms to solve the nonsymmetric eigenvalue problem in section 5. This leads us to use the following simple paradigm for using a highly parallel and occasionally numerically unstable algorithm:

(i) Solve the problem (quickly).
(ii) Test for instability (reliably)
(iii) If the answer is unsatisfactory, recompute the answer using a slower but more stable algorithm.

This paradigm will be successful if:

(i) The fast algorithm is only rarely unstable.
(ii) The instability test is cheap.

3. Dense Linear Algebra Libraries

In this section, we review the development of some important packages of dense linear algebra software.

3.1. EISPACK and LINPACK.

EISPACK [**63, 33**], released in 1972, is a collection of Fortran subroutines for solving matrix eigenvalue problems. In addition, two routines are included to compute the singular value decompositio and certain least-squares problems. The package is primarily based on a collection of Algol procedures developed in the 1960s and collected by Wilkinson and Reinsch [**72**]. Algorithms were chosen on the basis of their generality, elegance, accuracy, speed and economy of storage.

LINPACK [**20**], released in 1979, is a collection of Fortran subroutines that analyze and solve linear equations and linear least-squares problems. LINPACK is organized around the four matrix factorizations: LU, Cholesky, QR and singular value decomposition. The key factors influencing LINPACK's efficiency are the use of column-oriented algorithms and Level 1 BLAS (Basic Linear Algebra Subprograms) [**48**]. The column-oriented algorithms increase efficiency by preserving locality of reference (since Fortran stores arrays in column major order).

The EISPACK and LINPACK software libraries were designed for supercomputers used in the 1970s and early 1980s, such as the CDC-7600, Cyber 205 and Cray 1.

3.2. BLAS and LAPACK.

EISPACK and LINPACK have for many years provided high-quality portable software for linear algebra problems. However, on modern high performance computers, they often achieve only a small fraction of the peak performance of a machine. For example, in Table 1, we see that the performance (measured in megaflops[1]) of the LINPACK subroutine for computing the Cholesky factorization of a symmetric positive definite matrix is only at 40% of the peak performance of a processor element (PE) Cray C-90. When it runs on 16 processors, it is only at 3% of the peak performance. EISPACK and LINPACK are inefficient because their memory access patterns disregard the multilevel memory

[1]Megaflops = 10^6 or millions of floating point operations per second. Gigaflops = 10^9 or billions of floating point operations per second. Teraflops = 10^{12} or trillions of floating point operations per second. Since the number of floating point operations of an algorithm is fixed, the higher megaflops rate an algorithm is, the faster it is.

TABLE 1. Speed of LINPACK, BLAS and LAPACK on Cray C-90 in Megaflops

	1 PE	16 PEs
Maximum speed	952	15238
LINPACK (Cholesky, n=500)	387	479
BLAS 2 (Matrix-vector multiplication)	895	5900
BLAS 3 (Matrix-matrix multiplication)	898	13000
LAPACK (Cholesky, n=500)	785	5700
LAPACK (Cholesky, n=1000)	854	9800

hierarchies, thereby spending too much time *moving data* instead of doing useful floating point operations.

How can we sufficiently control data movement and achieve good vectorization and parallelism to obtain the levels of performance that those machines can offer? The answer is through the use of BLAS. BLAS is a key to transportability of the programs. There are three levels of BLAS:

Level 1 BLAS [**48**]**:** for vector-vector operations, such as $y \leftarrow \alpha x + y$.

Level 2 BLAS [**22**]**:** for matrix-vector operations, such as $y \leftarrow \alpha Ax + \beta y$.

Level 3 BLAS [**21**]**:** for matrix-matrix operations, such as $C \leftarrow \alpha AB + \beta C$,

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

One might well ask why we need to specify the higher level BLAS since they can obviously be decomposed into lower level simpler operations. The reason is that higher level BLAS offers much more opportunity to exploit locality and parallelizability than the lower level BLAS [**32**]. For example, in Level 2 BLAS, the number of floating point operations for the operation $y \leftarrow Ax + y$ is $2n^2$, the minimum memory references is $n^2 + 3n$ and their ratio q is 3. When the data are too large to fit in the top of the memory hierarchy, we wish to perform the most flops per memory reference to minimize data movement. In Level 2 BLAS, the ratio $q = 3$ gives an upper bound. It can achieve near-peak performance on many vector processors. For example, on the single processor of Cray C-90, it achieves 94% of the peak performance (see Table 1). However, on the multiprocessors, the performance is limited by this ratio. On 16 processors of Cray C-90, it only achieves 38.7%. This limitation can be overcome by Level 3 BLAS. The number