MASTER

TITLE: Implementing Techniques for Elliptic Problems on
Vector Processors

AUTHOR(S): B. L. Buzbee

SUBMITTED TO: Academic Press for publication in the Proceedings
of the Elliptic Problem Solvers Conference
held in Santa Fe June 30-June 2, 1980.
July

DISCLAIMER

This document contains information that is classified as CONFIDENTIAL. It is to be controlled in accordance with the provisions of the Atomic Energy Act of 1954, as amended, and the regulations promulgated thereunder. It is to be released only to those persons who are authorized to receive it.

University of California

By acceptance of this article, the publisher recognizes that the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes.

The Los Alamos Scientific Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy.

LOS ALAMOS SCIENTIFIC LABORATORY

Post Office Box 1663 Los Alamos, New Mexico 87545

An Affirmative Action/Equal Opportunity Employer

IMPLEMENTING TECHNIQUES FOR ELLIPTIC PROBLEMS ON VECTOR PROCESSORS

B. L. Buzbee

Computer Science and Services Division
Los Alamos Scientific Laboratory
Los Alamos, New Mexico

1. INTRODUCTION

To provide the arithmetic power required by large-scale numerical simulations, the fastest computers today (for example, the Cray-1 and the Cyber 205) incorporate vector processing. In such computers, a vector is an n-tuple of numbers systematically stored in memory. Because these computers attain their highest level of performance when processing vectors, considerable effort has been spent in the past decade developing algorithms that can be expressed as operations on vectors.

In Section II of this paper we define two types of vector architecture. We then discuss the variation in performance that can occur on a vector processor as a function of algorithm and implementation, the consequences of this variation, and the performance of some basic operators on the two classes of vector architecture. We also discuss the performance of some higher level operators that should be used with caution. In Section III we review the implementation of techniques for elliptic problems using the operators discussed in Section II.

Included are Fast Poisson solvers, dissection, and point, line, block, and conjugant gradient schemes. In Section IV we note some areas of research.

11. VECTOR PROCESSORS

Vector Algorithms

A question that arises when considering work on vector algorithms is "Will recent developments in Josephson Junction (JJ) technology preempt further work on vector algorithms?" Figure 1 displays the execution bandwidth in scalar mode of supercomputers for the past 30 years. (We define a supercomputer to be the fastest computer available at any point in time.) This data shows the diminishing growth rate in scalar performance of supercomputers. It further suggests that there is an upper bound on the scalar performance of the computer.

FIGURE 1. Execution bandwidth of supercomputers.

If such a bound exists, its precise value is yet to be determined. Nevertheless, this data is consistent with projected performance of computers based on JJ technology. For example, Robinson projects that JJ technology will produce scalar processors with a cycle time of 2-5 nanoseconds (1); Matisso projects 3 nanoseconds (2). Because the cycle time of the Cray-1 is about 12-1/2 nanoseconds, JJ technology does not provide a quantum jump in performance. Further, because some three-dimensional calculations require a two order of magnitude increase in performance over the Cray-1 (3), JJ technology will not provide scalar processors with processing power sufficient for future needs. Provision of that power will probably be based on current trends to parallelism, and vector processing is one form of parallelism. Thus we anticipate a continuing need for, and interest in, algorithms that can exploit vector and multiprocessing capability.

Vector Architectures

Although a satisfactory taxonomy for vector architectures is yet to be devised, we define two classes of vector architecture: memory-to-memory (MM) and register-to-register (RR).

MM ARCHITECTURE. Typically, for an MM architecture to achieve its highest level of performance, it must process algorithms that satisfy the following boundary conditions.

- Operand and result vectors must be stored contiguously in memory; that is, successive elements of the vector must be stored in adjacent memory locations.
- Vectors must be long.

Examples of MM architecture are the CDC 7600 when processing vectors from large-core memory, the Texas Instrument ASC, and the CDC Cyber 200 series.

RR ARCHITECTURE. RR architecture typically involves some sort of cache between memory and the processing units, with arithmetic operations being performed on contents of the cache. These processors achieve their highest level of performance when processing algorithms that satisfy the following conditions:

- Data memory between memory and the cache is minimized.
- Parallel execution of the functional units is maximized.

Examples of this architecture are the 120B array processor of Floating Point Systems, the Cray-1 of Cray Research, Inc., and the BSP of Burroughs Corporation.

Performance of Vector Processors

Performance on a vector processor can vary widely as a function of algorithm and implementation. For example, when solving dense linear systems of equations, Cray-1 performance varies as follows.

<u>Mode of Implementation</u>	<u>Performance Level</u>
Scalar Fortran	2-6 megaflops ^a
Vector Fortran	2-30 megaflops
Vector Assembly Language	2-120 megaflops

^aMegaflop = one million floating point operations per second.

The consequences of this variation are substantial, because on any computer

$$\text{Execution Time} = N \cdot T,$$

where N is the number of operations and T is the average time per operation.

On scalar computers there is very little variation in T . Thus, throughout the age of electronic computation, we have focused on minimizing N , i.e., optimal complexity. However, as we have just noted, there is a large variation in T on vector processors. In devising vector algorithms, therefore, we must minimize the product $N \cdot T$. Consequently, on vector processors we may find it desirable to use algorithms that are nonoptimal in complexity, provided we can do the additional work at a sufficiently high rate (4,5). For example, a numerical simulation at the Los Alamos Scientific Laboratory requires the complimentary error to be evaluated millions of times. This function can be nicely approximated by the inverse of a sixth degree polynomial raised to the fourth power. Direct evaluation of this approximation on a scalar computer encompassed significant amounts of arithmetic operations. Thus, on scalar computers, it is tabulated and evaluation of the function is through interpolation on that table. This results in a small number of arithmetic operations but requires table lookup. On a vector processor, table lookup is relatively slow, and the faster thing to do is to evaluate the polynomial approximation. Although N is large, $N \cdot T$ is small.

The class of algorithms that minimize $N \cdot T$ is probably larger than the class that minimize N . Further, the variation in performance of vector processors as a function of vector

length often leads to polyalgorithmic software. Although we have a large "forest" in which to seek algorithms, care must be used. A consistent algorithm is defined to be one that is optimal in arithmetic complexity. A little thought shows that for sufficiently large problems a consistent algorithm will always outperform an inconsistent algorithm, irrespective of the mode of implementation and computer architecture. Conversely, given two consistent algorithms on a vector processor, one cannot settle their performance by studying the coefficient of the low-order term in their complexity. In this case, the question is which one produces the smaller execution time.

Basic Operators

The following basic operators are available in either hardware or software for MM and RR architectures.

1. Vector Diadics
2. Inner Product
3. Matrix Multiply
4. Polynomial Evaluation

A discussion of matrix multiply and polynomial evaluation for RR architecture can be found in Ref. 6. Matrix multiplication is a high-performance operation on vector processors even for banded matrices; for example,

$$\begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \emptyset \\ \cdot & d_1 & d_2 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \emptyset & \cdot & \cdot & \cdot & \cdot \end{pmatrix} x = d_1 \begin{pmatrix} x \\ \cdot \\ \cdot \\ \cdot \\ \cdot \end{pmatrix} + d_2 \begin{pmatrix} \cdot \\ x \\ \cdot \\ \cdot \\ \cdot \end{pmatrix}, \quad (1)$$

where $\tilde{x}$ is the vector x with the first element deleted (7).
The following high-level operators also perform well on both architectures.

- Single FFT (Refs. 5, 8)
- Families of linear system solvers, for example, FFTs (Refs. 5, 8); tridiagonals (Ref. 6).
- Banded system solvers (Refs. 6, 9).

We have distinguished families of FFTs from a single FFT because the performance for a family is significantly higher than for a single FFT.

"Judicious Use" Operators.

The following operators should be used with caution on either architecture:

- Recursions
- Table lookup
- Conditionals
- One-to-many mappings
- Many-to-one mappings

Caution is required with these operators because their overall performance on a vector processor will be poor if they are not combined with a sufficient amount of arithmetic. Included in recursions are solutions of a single tridiagonal system, which generally does not perform well on a vector processor (6, 10). Table lookup has implications for adaptive procedures. Poor performance of conditionals implies that we must forego our frequently used pointwise convergence test. See Ref. 11 for a discussion of programming considerations in the use of a vector processor.

III. IMPLEMENTING TECHNIQUES FOR ELLIPTIC PROBLEMS

In this section we discuss the implementation of classes of techniques for elliptic problems using the aforementioned operators. A similar discussion was given by Ortega and Voight (12).

FAST POISSON SOLVERS. Fast Poisson solvers are implementable on both MM and RR architectures by using operators to perform families of FFTs and to solve families of tridiagonals. On MM architecture, some care is required to ensure that the vectors manipulated are stored contiguously in storage. Workers in this field include Buzbee (13) and Kascic (14).

DISSECTION SCHEMES. One-way dissection schemes are implementable in any environment where there is good capability for solving small, dense, linear systems. Incomplete nested dissection is implementable on both types of architecture and is attractive if only a few dissections are done. Workers in this area include George, et al. (15), Voight (9), and Calahan (16).

POINT SCHEMES. Point Jacobi schemes are attractive on either class of architecture because they are implementable by vector diadics and banded-matrix multiply operators involving long vectors (17). These advantages have prompted renewed interest in these techniques (18) with an objective of producing Jacobi techniques with attractive convergence rates.

Point successive overrelaxation schemes can be implemented on either architecture with wave fronting on a natural ordering or by checkerboard ordering. Wave fronting involves processing the mesh by diagonals. Because vectors must be stored

contiguously on MM architectures, this is not an attractive alternative on them. It is feasible on RR architecture. Using the checkerboard ordering, implementation on either architecture is feasible if the red and black points are stored as separate arrays (19, 20). It follows that multigrid schemes are implementable on either class of architecture. Of course, these techniques involve one-to-many and many-to-one mappings. These mappings occur in families and can be vectorized.

LINE SCHEMES. Line schemes are implementable by using operators for solving families of tridiagonal systems. This is true for both line relaxation schemes and alternating direction schemes. In the latter case some care must be used on MM architecture to produce vectors contiguously stored in memory. For example, Gaussian elimination can be used in one coordinate direction and odd-even reduction in the other to produce contiguously stored vectors (21). On RR architecture, line schemes are easily implemented in Fortran and achieve high rates of vectorization (22).

BLOCK SCHEMES. Block schemes are easily implemented on both architectures using operators to solve banded linear systems or block tridiagonal systems. Unfortunately they suffer from excessive arithmetic complexity and storage requirements. Progress is being made toward reducing their complexity (23).

CONJUGANT GRADIENT SCHEMES. Conjugant gradient techniques can be implemented on vector processors using any of the previously discussed schemes as the approximate factorization. Also, incomplete Cholesky vectorization can be implemented on either class of architecture. On MM architecture odd-even

reduction has been used (17), and on RR architecture a variant of block elimination is used (24).

IV. RESEARCH ISSUES

We noted that matrix multiplication is an efficient operator on vector processors even in the banded case because the operator can be expressed in terms of vector diadics on the diagonals of the matrix. This raises the question of whether one can solve linear systems of equations by manipulating diagonals. Such an algorithm exists for tridiagonal systems (25), but not for the general banded case. If available, such techniques would be attractive, especially on MM architecture.

Block schemes are attractive on vector processors because of their ease of implementation and their amenability to vectorization. Faber (23) has shown that certain strategies of block relaxation produce high convergence rates. The problem remaining is to produce efficient techniques for dealing with the associated blocks.

Three-dimensional problems are occurring with increasing frequency. Typically, the associated mesh is held in secondary storage of some sort, with the result that overall execution time is a function of how often the mesh must be transmitted to and from the central processor. This suggests development of techniques that minimize the number of iterations, perhaps at the expense of additional arithmetic operations.

REFERENCES

1. A. L. Robinson, "Superconducting Electronics; Toward an Ultrafast Computer," Science 201 (August 1978).
2. J. Matisoo, "The Superconducting Computer," Scientific American (May 1980).
3. B. L. Buzbee, et al., "DOE Research in Utilization of High-Performance Computers," to appear as a Los Alamos Scientific Laboratory Report.
4. N. K. Madsen and G. H. Rodrigue, "Two Notes on Algorithm Design for the CDC-STAR-100," Numerical Mathematics Group Informal Technical Memorandum 75-1, Lawrence Livermore Laboratory (July 1975).
5. O. G. Korn and T. T. Lambiotte, Jr., "Computing the Fast Fourier Transform on a Vector Computer," Reprint No. 78-5, ICASE, NASA Langley Research Center (February 1978).
6. K. Fong and T. L. Jordan, "Some Linear Algebraic Algorithms and Their Performance on Cray-1," Los Alamos Scientific Laboratory Report LA-6774 (June 1977).
7. T. I. Karush, N. K. Madsen, and G. H. Rodrigue, "Matrix Multiplication By Diagonals on Vector/Parallel Processors," Lawrence Livermore Laboratory report UCID-16899 (August 1975).

8. C. Temperton, "Fast Fourier Transforms on Cray-1," European Centre for Medium Range Weather Forecasts Report 21, (January 1979).
9. R. G. Voight, "The Influence of Vector Computer Architecture on Numerical Algorithms," ICASE, NASA Langley Research Center, Report 77-8 (March 1977).
10. N. K. Madren and G. H. Rodrigue, "A Comparison of Direct Methods for Tridiagonal Systems on the CDC-STAK-100," Lawrence Livermore Laboratory report NCRL-76993 (July 1975).
11. B. Brode, "How to Get More Out of Your Vector Processor," Proceedings of the 1978 LASL Workshop on Vector and Parallel Processors, Los Alamos Scientific Laboratory report LA-7491-C (September 1978).
12. J. M. Ortega and R. G. Voight, "Solution of Partial Differential Equations on Vector Computers," ICASE, NASA Langley Research Center, report 77-7 (March 1977).
13. B. L. Buzbee, "A Fast Poisson Solver Amenable to Parallel Computation," IEEE Transactions on Computers C-22 (August 1973).
14. M. T. Kascic, Jr., "A Direct Poisson Solver on STAK," Proceedings of the 1978 LASL Workshop on Vector and Parallel Processors, Los Alamos Scientific Laboratory report LA-7491-C (September 1978).

15. A. George, W. G. Poole, Jr., and K. G. Voight, "Analysis of Dissection Algorithms for Vector Computers," ICASE, NASA Langley Research Center, Report 76-17 (June 1976).
16. D. Calahan, "An Equation Solver Vectorized for Both Simultaneous Sparse and Single Dense Systems, these proceedings.
17. C. Hendrickson, M. Pratt, and G. Rodrigue, "The Numerical Solution of the Lagrangian Diffusion Equation on a Vector Processor," Lawrence Livermore Laboratory report UCRL-82430 (June 1979).
18. O. Johnson and G. Paul, "Vector Algorithms for Elliptic P.D.E.s Based On the Jacobi Method," these proceedings.
19. B. Buzbee, G. Golub, and J. A. Howell, "Vectorization for the Cray-1 of Some Methods for Solving Elliptic Difference Equations," Proceedings of the Symposium on High Speed Computer and Algorithm Organization, Urbana, Illinois (April 1977).
20. W. Gautschi, C. Weiland, and D. Muller-Wichard, "Possibilities and Problems with the Application of Vector Computers," German Research and Testing Establishment for Aerospace (April 1980).
21. P. Duboin and G. Rodrigue, "Operator Splitting on the STAR Without Transposing," Lawrence Livermore Report UCID-17515 (June 1977).

22. D. Boley, B. L. Buzbee, and S. V. Parter, "Applications of Block Relaxation," Proceedings of AFM Symposium on Reservoir Simulation (February 1979).
23. V. Faber, "Block Relaxation Strategies," these proceedings.
24. T. L. Jordan and M. Mahaffy, private communication, 1980.
25. T. L. Jordan, "A New Parallel Algorithm for Diagonally Dominant Tridiagonal Matrices," Los Alamos Scientific Laboratory report LA-UR-74-1903 (1974).