LA-UR -83-1761

CONF-831111--3

MASTER

LA-UR--83-1761

DE83 014125

TITLE VECTORIZATION OF ALGORITHMS FOR SOLVING SYSTEMS OF ELLIPTIC DIFFERENCE EQUATIONS

AUTHOR(S) B. L. Buzbee

SUBMITTED TO ASME Symposium on Impact of New Computing Systems on Computational Mechanics

Boston, Massachusetts

November 13-18, 1983

Los Alamos Los Alamos National Laboratory
Los Alamos, New Mexico 87545

FORM NO 836 R4
ST NO 2629 5/81

VECTORIZATION OF ALGORITHMS FOR SOLVING SYSTEMS OF ELLIPTIC DIFFERENCE EQUATIONS

B. L. Buzbee
Computing Division
Los Alamos National Laboratory
Los Alamos, New Mexico

ABSTRACT

Today's fastest computers achieve their highest level of performance when processing vectors. Consequently, considerable effort has been spent in the past decade developing algorithms that can be expressed as operations on vectors. In this paper we define two types of vector architecture. We discuss the variation of 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 higher level operators, including some that should be used with caution. Using both basic and high level operators, we discuss vector implementation of techniques for solving systems of elliptic difference equations. Included are fast Poisson solvers and point, line, and conjugate gradient techniques.

I. 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, we will discuss vectorization of algorithms for solving systems of elliptic difference equations.

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 solving systems of difference equations using the operators discussed in Section II. Included are Fast Poisson solvers and point, line, block, and conjugant gradient schemes. Section IV summarizes the results.

II. VECTOR PROCESSORS

Are Vector Algorithms Obsolete?

Recent developments have caused many people to ask if Josephson Junction (JJ) technology will eliminate the need for vector processors? Unfortunately, the answer is negative. Figure 1 displays the execution bandwidth in scalar mode of supercomputers for the past 40 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 u[illegible] bound on the scalar performance of computers and this data is consistent with projected [illegible]lar performance of computers built with JJ technology. For example, Robinson projects th[illegible] JJ technology will produce scalar processors with a cycle time of 2-5 nanoseconds (1); Matisoo pro[illegible] to 3 nanoseconds (2). Because JJ technology will not be available until the late 1980s, it does not provide a quantum jump in performance relative to Fig. 1. Further, because some 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 require incorporation of JJ technology into v[illegible]ctor processors, and perhaps multiple vector processors. Thus we anticipate a continuing need for, and interest in, algorithms that can exploit vector and multiprocessing capability.

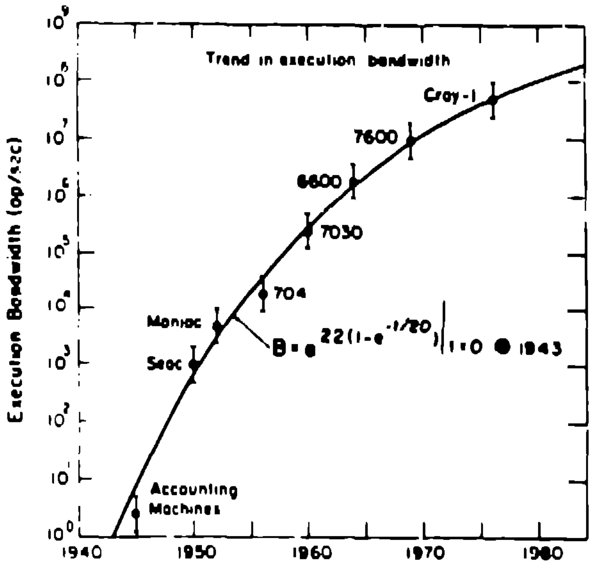

Fig. 1. Execution bandwidth of supercomputers.

Performance as a Function of Vectorization

A vector processor is a two-state machine. In the scalar state, it is relatively slow, but general purpose. In the vector state it is relatively fast on vector operations. For a given application, let speedup be defined as

$$S_\rho = \frac{\textit{execution time in scalar mode}}{\textit{execution time after vectorization}} .$$

Assume that vector mode is ρ times faster than scalar mode. Let

α = *fraction of instructions that can be vectorized.*

If we normalize the scalar execution time to unity, then

$$S_\rho = \frac{1}{(1-\alpha) + \frac{\alpha}{\rho}} .$$

Figure 2 shows overall performance as a function of α for three choices of ρ. Clearly "a little vectorization does not go far." High performance from a vector processor necessitates a high percentage of vectorization.

The above expression for S_p assumes that the same number of instructions are executed in both scalar and vectorized implementations. Seldom is this the case, so in practice S_p is even less favorable than indicated above.

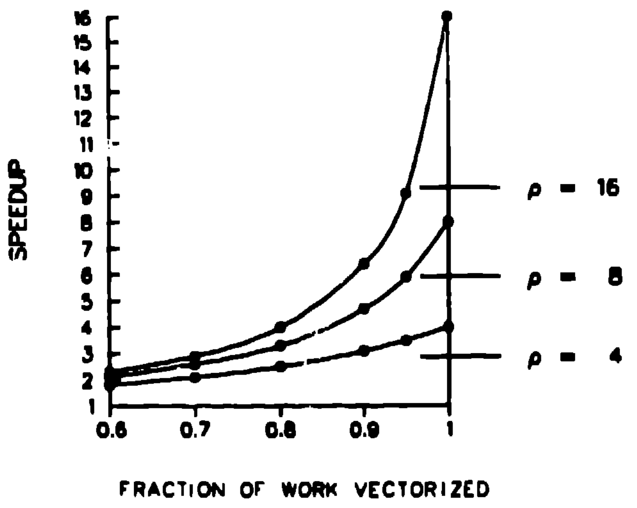

Fig. 2. Speedup as a function of vectorization and vector performance relative to scalar performance.

Impact of Vector Processors on Algorithms

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:

Mode of Implementation	Performance Level
Scalar Fortran	2-6 megaflops[a]
Vector Fortran	2-30 megaflops
Vector Assembly Language	2-120 megaflops

[a] *megaflop = one million floating point operations per second.*

As we will see, the consequences of this variation are substantial because on any computer

Execution Time = N·T ,

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

On scalar computers there is little variation in T. Thus, throughout the age of electronic computation, we have focused on minimizing N, i.e., optimal complexity. There is a large variation in T on vector processors, so in devising vector algorithms we must minimize the product N·T. Consequently, 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 Los Alamos 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 encompasses significant amounts of arithmetic operations. On scalar computers, it is tabulated and the function is evaluated 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 alternative is to evaluate the polynomial approximation. So, although N is large, N*T is small.

The class of algorithms that minimizes N*T is probably larger than the class that minimizes N. Thus, we have a larger "forest" in which to seek algorithms. However, care must be exercised. 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. Also, given two consistent algorithms on a vector processor, we cannot settle their performance by studying the coefficients of the low-order term in their complexity. The question is which one produces the smaller execution time as a function of vector length. Because of the variation in vector processor performance as a function of vector length and mode of implementation, we often encounter polyalgorithmic software on vector processors.

Vector Architectures

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, 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 movement 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 and the Cray-1 of Cray Research, Inc.

Basic Operators

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

Vector Dyadics
Inner Product
Matrix Multiply
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,

$$\begin{pmatrix} \ddots & \ddots & & \\ & d_1 & d_2 & \\ & & \ddots & \ddots \\ & & & \ddots \end{pmatrix} x = d_1 \otimes x + d_2 \otimes \tilde{x} \quad .$$

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

- Single FFT (5,8).
- The same operator applied to sets of data; for example, sets of FFTs (5,8) and sets of tridiagonal systems (6).
- Banded system solvers (6,9).

We have distinguished sets of FFTs from a single FFT because the performance for a set 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 is the solution 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).

Point Schemes

Point Jacobi schemes are attractive on either class of architecture because they are implementable by vector dyadics and banded-matrix multiply operators involving long vectors (13). These advantages have prompted renewed interest in these techniques (14) 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 (15,16). 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 sets and can be vectorized.

Line Schemes

Line relaxation schemes are implementable by using operators for solving sets of tridiagonal systems combined with an odd/even ordering of lines. Alternating direction schemes are implementable using operators for solving sets of tridiagonals. In the latter case some care must be used on MM architecture to produce vectors contiguously stored in memory. For example, Gaussian elimination on the tridiagonal can be used in one coordinate direction and odd-even reduction in the other to produce contiguously stored vectors (17). On RR architecture, line schemes are easily implemented in Fortran and achieve high rates of vectorization (18).

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 (13), and on RR architecture a variant of block elimination is used (19).

Fast Poisson Solvers

Fast Poisson solvers are implementable on both MM and RR architectures by using operators to perform sets of FFTs and to solve sets 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 (20) and Kascic (21).

SUMMARY

High performance on a vector processor necessitates a high percentage of vectorization. High percentage of vectorization may require new data structures and new algorithms, even the use of nonoptimal algorithms. Perhaps surprisingly, suitable data structures and algorithms are available such that familiar schemes for solving elliptic systems of difference equations can be highly vectorized for either memory-to-memory vector processors or register-to-register vector processors.

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, W. J. Worlton, G. Michael, and G. Rodrigue, "DOE Research in Utilization of High-Performance Computers," Los Alamos National Laboratory report LA-8470-MS (1980).

(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 National 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 National 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 National Laboratory report UCID-16899 (August 1975).

(8) C. Temperton, "Fast Fourier Transforms on Cray-1," European Centre for Median 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. Madsen and G. H Rodrigue, "A Comparison of Direct Methods for Tridiagonal Systems on the CDC-STAR-100," Lawrence Livermore National 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 National Laboratory report LA-7491-C (September 197F

(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) C. Hendrickson, M. Pratt, and G. Rodrigue, "The Numerical Solution of the LaGrangian Diffusion Equation on a Vector Processor," Lawrence Livermore National Laboratory report UCRL-82930 (June 1979).

(14) O. Johnson and G. Paul, "Vector Algorithms for Elliptic P.D.E.s Based On the Jacobi Method," Proceedings, Elliptical Problem Solvers Conference, Santa Fe, NM, June 1980 (Academic Press, 1980).

(15) 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).

(16) W. Gautzsch, C. Weiland, and D. Muller-Wichards, "Possibilities and Problems with the Application of Vector Computers," German Research and Testing Establishment for Aerospace (April 1980).

(17) P. Dubois and G. Rodrigue, "Operator Splitting on the STAR Without Transposing," Lawrence Livermore National Laboratory report UCID-17515 (June 1977).

(18) D. Boley, B. L. Buzbee, and S. V. Parter, "Applications of Block Relaxation," Proceedings of AIME Symposium on Reservoir Simulation (February 1979).

(19) T. L. Jordan and M. Mahaffy, private communication, 1980.

(20) B. L. Buzbee, "A Fast Poisson Solver Amenable to Parallel Computation," *IEEE Transactions on Computers C-22* (August 1973).

(21) M. T. Kascic, Jr., "A Direct Poisson Solver on STAR," Proceedings of the 1978 LASL Workshop on Vector and Parallel Processors, Los Alamos National Laboratory report LA-7491-C (September 1978).

(22) T. L. Jordan, "A New Parallel Algorithm for Diagonally Dominant Tridiagonal Matrices," Los Alamos National Laboratory report LA-UR-74-1903 (1974).