CONF-781061--1

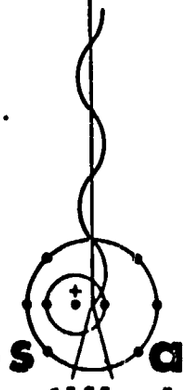
LA-UR-78-2691

MASTER

TITLE: NUMERICAL ALGORITHMS AND SOFTWARE FOR ADVANCED COMPUTERS

AUTHOR(S): B. L. Buzbee, C-3

SUBMITTED TO: To be presented and published in the proceedings at the AESOP-SCIE Symposium on Advanced Computing, October 19, 1978, Sands Hotel, Las Vegas, Nevada



los alamos
scientific laboratory
of the University of California
LOS ALAMOS, NEW MEXICO 87545

Form No. 836
St. No. 2629
1 75

UNITED STATES
ENERGY RESEARCH AND
DEVELOPMENT ADMINISTRATION
CONTRACT W-7405-ENG. 36

NUMERICAL ALGORITHMS AND SOFTWARE FOR ADVANCED COMPUTERS

by

B. L. Buzbee

ABSTRACT

This report discusses the utilization of large-scale computers at Los Alamos Scientific Laboratory and why scientists are constantly seeking bigger and faster computers. We also follow the trend toward increased parallelism within the architecture of supercomputers and show how this parallelism is affecting software and algorithms. Based on this trend and characteristics of existing simulation models, we indicate some of the areas where future research will be needed.

I. INTRODUCTION

Although current generations of supercomputers can perform tens of millions of arithmetic operations per second, they are still incapable of doing all of the numerical simulations that scientists would like to do. In this report we discuss the utilization of supercomputers at the Los Alamos Scientific Laboratory, and provide some insight into why scientists are continually seeking larger and faster computers. We also follow the trend toward increased parallelism in the architecture of supercomputers and show how this parallelism is affecting algorithms and software. Based on this trend and characteristics of present day simulation models we indicate some future directions of research.

II. LARGE-SCALE COMPUTATION AT LASL

Large-scale scientific computations are those computational problems that saturate all resources of any available computer. Examples of such problems include aerodynamic simulation, nuclear weapon design, atmospheric modeling, biological modeling, economic modeling, etc. Computers that are best suited for performing

large-scale computations are frequently referred to as supercomputers. The associated computing systems are generally characterized by

- heavy usage of FORTRAN,
- hardware that is very fast in performing floating point operations,
- large memory capacities,
- large mass storage facilities,
- interactive graphics,
- state-of-the-art technology.

One can also define supercomputers by analogy. For example, Edward Yourdon [1] attributes four characteristics to the superprogrammer:

1. They are freaks in some sense of the word.
2. They refuse to work regular hours.
3. They tend to use relatively few software tools.
4. They do not get along well with people.

It is remarkable that a recent LASL document describes supercomputers as follows:

1. They incorporate state-of-the-art technology (they are freaks in some sense of the word).
2. They are relatively unreliable (refuse to work regular hours).
3. They tend to be software poor (use relatively few software tools).
4. There are relatively few of them (do not get along well with people).

So if you know something about superprogrammers, you, in fact, know something about supercomputers.

LASL was established in 1943 to develop a nuclear weapon. Since that time, its activities have diversified, but weapons design continues to be a major effort and dominates the usage of supercomputers. This activity accounts for 60-70% of LASL supercomputer utilization. The other 30-40% is used by energy research projects, such as fusion energy studies and fission reactor safety

studies. A recent a pical weapon design required approximately 6000 hours of CDC equivalent processor time, which is nearly 1 1/2 years' co nuous operation. That is an enormous amount of computation, at the cost of it is substantially less than the total system cost. Because use of numerical computation reduces the number of weapons tests, accelerates the design process, etc., acquisition of supercomputers is, and will continue to be, justified.

Numerical simulation performs at least three important functions for the design process. One function is the study of fine structure, that is, study of local behavior whose experimental observation is at best difficult, perhaps impossible. Such things as the behavior in time at the interface of two materials is an example. Another function is sensitivity, that is, determining charge in performance as a function of design parameter variation. Finally, there is the function of optimization--determining optimum performance as a function of design parameters. Each of these functions requires many iterations, and it is through those iterations that thousands of hours of computation are accumulated.

Several aspects of numerical simulation have important implications for computing facilities.

- Different codes are used to represent overlapping phases of device function. As a consequence, a family of programs must access a common data base. This leads to a requirement for large mass-storage facilities.

- There is an upper limit on the time that can be tolerated for execution of a design code. "Overnight" defines the largest practical time available for a given iteration. If execution time exceeds this upper limit, the design process begins to suffer because of inability to meet design schedules, machine failure, and because the designer is unable to manage several computations concurrently. This upper limit varies from 10 hours to 2 days at LASL. A current design code contains about 90,000 lines of FORTRAN, requires 67 overlays for execution on a CDC 7600, and needs 10 million words of disk storage for its execution. It requires 9 hours of central processor (CP) time for its execution, which meets the "overnight" criterion, and it performs approximately 100 billion floating point operations during that 9 hours. An improved model requires 30 to 40 hours of execution time, and that, of course, exceeds the "overnight" criterion. It needs about 200 million words of disk storage and will perform approximately 400 billion floating point operations during this execution. Consequently, the requirement for "overnight" execution leads us to continually seek bigger and faster computers. Improvements to the models include increased accuracy in space and time as well as more and better physics.

- Finally, many design codes require interaction with the designer during their execution, and this leads to the requirement for good interactive graphics capability.

Our ability to design is directly proportional to the capability and capacity of advanced computers. Thus, further increases in computer performance will be used to improve models rather than reduce computational time and cost.

III. EVOLUTION OF SUPERCOMPUTER PERFORMANCE AND ARCHITECTURE

The trend of execution bandwidth in supercomputers during the past 25 years is shown in Fig. 1. Computers in the early 50's performed a few thousand operations per second, whereas recent supercomputers perform 10 to 100 million floating point operations per second. Thus within the past 25 years arithmetic performance has increased by five orders of magnitude. These data are nicely approximated by a double exponential in time that is beginning to flatten out. We will have more to say about that later.

Figure 2 shows growth of memory capacity. Comparing the Maniac 1 with the CRAY-1, memory capacity has increased by some three orders of magnitude.

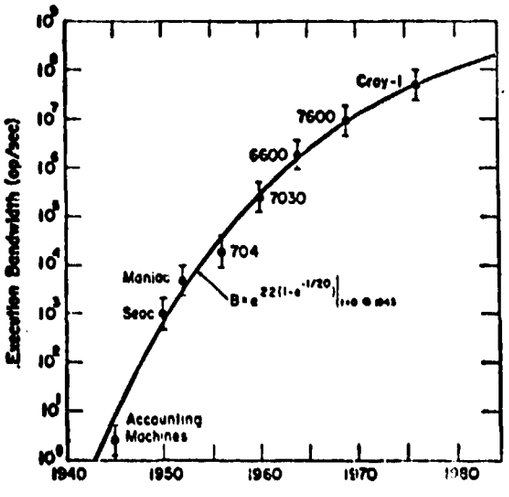


Fig. 1.
Trend in execution bandwidth.

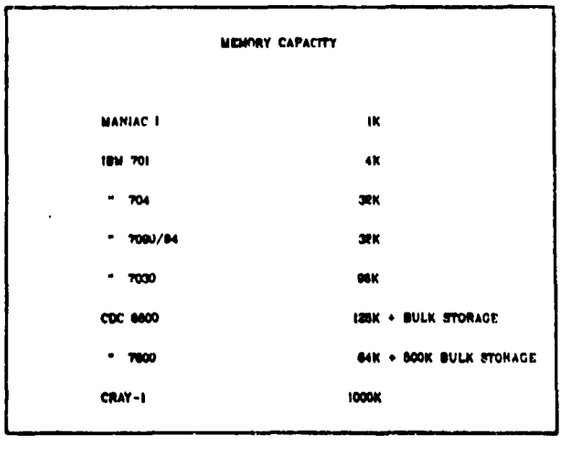


Fig. 2.
Growth in memory capacity.

There is a definite trend within computer architecture towards parallelism. Computers in the early 50's were strictly sequential in their issue and execution of instructions. That is, the instruction was fetched from memory, decoded, its operands were fetched, the operation performed, and the result stored. Then the next instruction was fetched, decoded, etc. In the early

60's computers appeared in which the execution of instruction (n) was overlapped in time with the fetching and decoding of instruction (n + 1). In the mid 60's computers had independent functional units. For example, the CDC 6600 had independent units for performing multiplication, addition, etc., and those units were capable of parallel execution. In the late 60's computers had enough parallelism in them to achieve vector mode. By vector mode we mean that at any time the computer may be processing two or more results. To achieve this mode, we need parallelism in memory, obtained by interleaving independent banks of memory; an instruction stack in the CPU to suppress instruction fetching so that operand streaming is not disrupted; multiple registers in the CPU to hold intermediate results; and, finally, segmented functional units to sustain the stream through the CPU. Computers such as the CDC 7600 and the IBM 360/195 have sufficient features to achieve this mode. In the early 70's computers appeared with explicit vector instructions, hardwired instructions that operate on n-tuples of numbers. The early machines were memory-to-memory in that the vector instruction was performed on vectors stored in memory and the result vector was stored in memory. Recent vector computers perform instructions on the contents of vector registers in the CPU and then store the results in a CPU vector register. Examples of memory-to-memory architecture are the Texas Instruments ASC and the CDC Star-100; the Cray Research Inc., (CRI) CRAY-1 is an example of register-to-register architecture. Finally, as of 1977, the DAP (Distributed Array Processor) array of processors is available from International Computer, Limited, England. Of course, arrays of processors have been operational for several years, but this is the first commercially available product. Thus there has been a growing trend toward increased parallelism in architecture. We will also come back to this point.

IV. IMPACT OF SUPERCOMPUTER ARCHITECTURE ON ALGORITHMS AND SOFTWARE

Beginning with the introduction of the Star-100, the architecture of supercomputers has had substantial impact on algorithms and software. Although computers such as the CDC 7600 had parallelism in their architecture, the average user could get a significant gain in performance from the CDC 7600 relative to its predecessors without any knowledge of the parallelism in it. As evidenced by the following quotations, such is no longer the case.

> "NOBODY, AND I MEAN NOBODY, KNOWS HOW TO PROGRAM LARGE PARALLEL MACHINES" - Seymour Cray, Business Week, 12/6/76.

> "MANY CURRENT LARGE-SCALE PROGRAMS CANNOT RUN EFFICIENTLY ON ANY KNOWN HIGH PERFORMANCE MACHINE..." - David W. Hogan, John C. Jensen and Merrill Cornish, Texas Instruments, Inc.

"CONVERTING PROBLEMS FOR EFFICIENT EXECUTION ON THE NEW CLASS OF 'HIGHLY PARALLEL' MACHINES REQUIRES AN ALMOST COMPLETE REANALYSIS AND REWRITE OF THE PROGRAM." - J. E. Wirsching and T. Kishi.

"...ALGORITHMS FOR THE COMPUTERS MENTIONED ABOVE MUST BE DEVELOPED WITH THE SPECIFIC ARCHITECTURE WELL IN MIND IF ANYTHING APPROACHING THE FULL POTENTIAL OF THESE MACHINES IS TO BE REALIZED." - Robert G. Voigt, ICASE.

"AT LLL WE CONSERVATIVELY EXPERIENCE A TEN MAN YEAR EFFORT PER APPLICATION CODE TO ACHIEVE A SPEED UP OF 2 - 3 ON THE CDC STAR-100 OVER A CDC-7600. THIS SHOULD BE CONTRASTED TO A NOMINAL EFFORT (MONTHS) FOR A FACTOR OF FIVE IMPROVEMENT IN THE CODE MIGRATION FROM THE CDC-6600 TO THE CDC-7600." - T. Rudy, LLL.

The integral of these statements shows that many existing simulation models do not perform efficiently on new supercomputers and that in order to make them do so, the models must be reprogrammed with careful attention to the architecture and in some cases algorithms must be replaced. Furthermore, the total effort required to do this is greater than what we are accustomed to expending on the conversion of programs from one generation of computers to another.

A recent computational experiment at LASL provides quantitative data in support of the foregoing quotations. Table I summarizes the results of this experiment in which a single algorithm was implemented on the CRAY-1 in eight different environments. The parameter displayed in Table I is millions of floating point operations per second (megaflops); thus, the higher the number, the better the performance. Data for the first seven environments are from [2]. Datum for the eighth is from [3].

TABLE I

MILLIONS OF FLOATING POINT OPERATIONS/SECOND (MEGAFLOPS) ACHIEVED IN DIFFERENT ENVIRONMENTS

	SGEFA
1. FTNX Inline	1.3
2. CRI Inline VECT=OFF	3.3
3. CRI FORTRAN BLAS	4.5
4. FTNX FORTRAN BLAS	6.0
5. FTNX Vectorized Inline	16.6
6. CRI Inline VECT=ON	23.8
7. FTNX CAL BLAS	27.4
8. Optimal CAL [4]	120.0

In the first environment, the algorithm was specified in FORTRAN without any calls to subordinate routines, i.e., "inline", and compiled by LASL's FTNX compiler with vectorization prohibited. Environment number two is identical to one, except that compilation was by the Cray Research, Inc., CFT compiler. In environment three, the innermost DO loop was replaced by a call to a subordinate routine. This routine is written in FORTRAN and performs the vector operation required by the innermost DO loop. The routine is written in a fashion that will achieve high performance on computers that have instruction stacks such as the CDC 7600. Unfortunately, this mode of specification does not vectorize on the CRAY-1. Environment three is this mode of implementation with compilation by the CRI compiler. Environment four is identical to environment three, except that compilation is by the FTNX compiler. Environment five uses the inline source from environment one with compilation by the FTNX compiler and vectorization wherever possible. Environment six is similar to five, except that compilation is by the CRI compiler. Environment seven is identical to environment three, except that the vector subroutine is hand coded in assembly language and uses vector operations. Finally, environment eight is a hand-coded assembly language implementation of the algorithm carefully formulated to match the CRAY-1 architecture.

Table I shows that the performance of a vector computer can vary widely as a function of software and algorithm specification. Because of this variation, every laboratory that has acquired a vector processor has had to restructure compute-bound sections of simulation models in order to achieve appropriate performance levels from these processors. This restructuring of code accounts for the relatively large conversion cost alluded to in the quotation by Rudy. Conversion cost must also include development of new algorithms. For example, in environment eight of Table I, the algorithm has been very carefully tailored to fit the architecture of the CRAY-1. In some situations, completely new algorithms must be developed. For example, the traditional equation of state table look-up procedure used at LLL does not vectorize. Research to develop a vectorizable algorithm was successful and yielded an algorithm that performs an order of magnitude better on the Star-100 than the traditional procedure[4].

V. CURRENT SIMULATION MODELS

Generally, the value of vector hardware applied to current simulation models is not nearly as great as suggested by Table I. To assess its value in the average situation, assume that we have two computers--one is a scalar computer and the other a vector computer. Of course, vector computers can perform scalar operations. So let C_s be the average time required by the scalar computer to perform a scalar operation and let C_v be the average time required by the vector computer to perform a scalar operation. Assume that the scalar speeds of the two computers are equal and

that the vector machine performs vector operations in zero time, that is, it is infinitely fast in vector mode. Figure 3 illustrates the performance of the vector computer relative to the scalar computer under these assumptions as a function of vectorization. If we can vectorize 50% of the total work in our computation, that work will be done in zero time on the vector computer and we will get a factor of 2 improvement from it relative to the scalar computer. Now change one of the assumptions. Assume that the scalar computer is faster and performs scalar operations in only one-fourth the time required by the vector computer. Figure 4 shows the relative performance. Notice that we must vectorize 75% of the total work just to break even. To really get a performance gain from the vector computer in this case, we must operate in the area of 95% vectorization. That is difficult to achieve. In general, it requires a complete rewrite of the program in assembly programming language. Assume now that the scalar computer is slower and that it takes twice as long to perform a scalar operation as the vector computer. Figure 5 shows the relative performance. We start with a factor of 2; if we can vectorize 50% of the work, we will have a factor of 4.

The CRAY-1 is about 2.5 times as fast as the CDC 7600 in scalar mode, and our experience is that compilers and other software packages can achieve 25-50% vectorization of the total work in existing models. Thus we believe the CRAY-1 will perform 3-5 times the CDC 7600.

VI. THE FUTURE

During the past 10 years, development of new simulation models in large-scale computation has been a secondary activity. This is easy to understand. If a scientist has a 40 or 50 thousand line FORTRAN program and wants it to run four times faster, there are two alternatives. One is to modify the program, optimizing it algorithmically. The other is to look for a computer that runs four times faster. In the past 25 years, the arithmetic performance of supercomputers has grown by five orders of magnitude, so the decision was easy. But things are changing. In the last decade performance has increased by only one order of magnitude and there is wide agreement that the growth rate will continue to diminish. Most people believe that scalar performance will not increase by more than an order of magnitude in the next 10 years, and some predict that it will grow by only a factor of 4. On the other hand as discussed in Section II, our demand for computation is unabated. We have barely scratched the surface in three-dimensional simulation and there is growing interest in it.

If computers are beginning to level out in their performance (in other words the exponential in Fig. 1 accurately predicts the future) and our demand for computation is unsatisfied, what shall we do? There are two possibilities. One is to use parallel computation. Vector computers are a form of parallel computation.

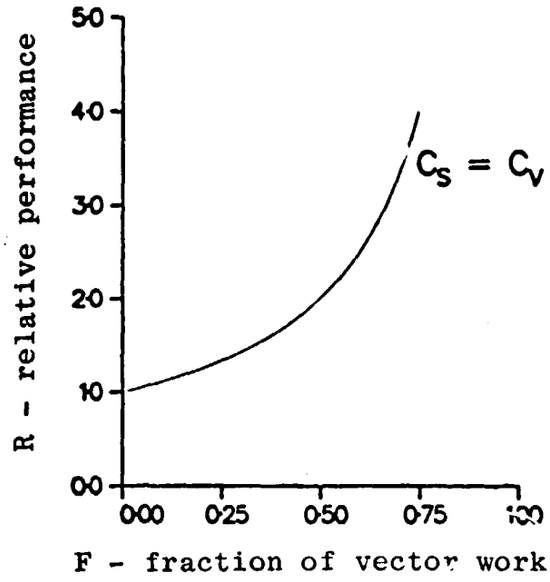


Fig. 3.
Relative performance when scalar speeds are equal.

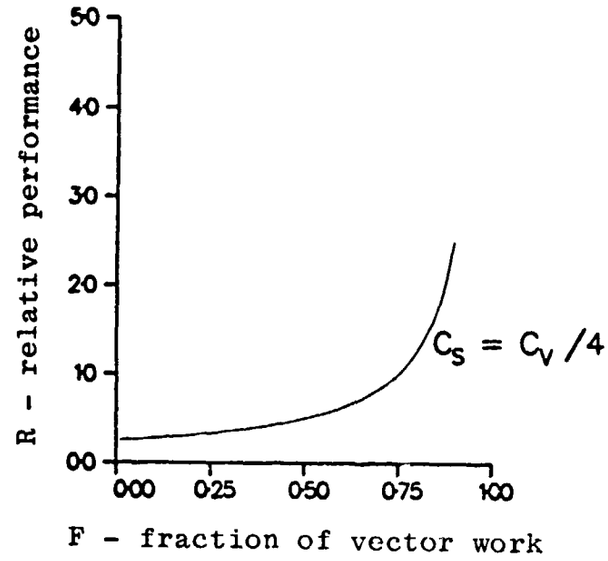


Fig. 4.
Relative performance when scalar computer performs scalar operations four times as fast as the vector computer.

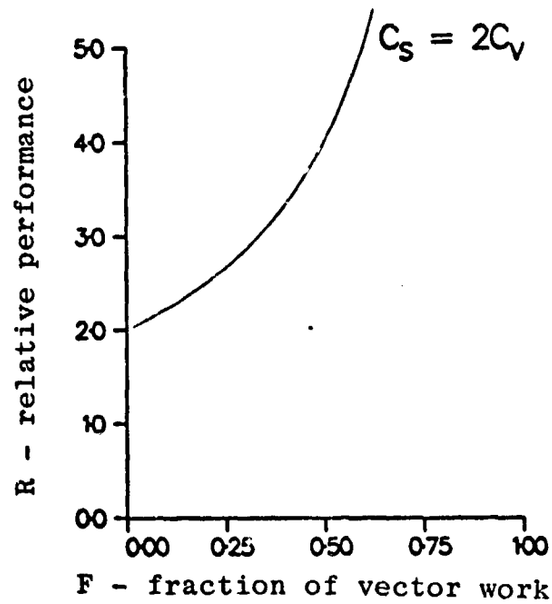


Fig. 5.
Relative performance when vector computer performs scalar operations twice as fast as the scalar computer.

But our present algorithms require a large amount of scalar computation and as long as we only achieve 30, 40, or 50% vectorization, scalar performance dominates the overall performance of the computer. To make vector computation really pay off, we must vectorize 95-99% of the total work. This means new algorithms must be developed. We may also use arrays of processors. But, again, making them perform well requires development of new algorithms. A second possibility is to use existing scalar computers but change the numerical techniques in our models. The National Aeronautics and Space Administration (NASA) recently commissioned two vendors to study the question of whether or not they could build a computer that would perform a billion floating point operations per second and have 40 million words of memory. They were led to those requirements because they are interested in three-dimensional simulation on a 100 by 100 by 100 mesh and they need 38 variables per mesh point. As people in NASA have pointed out, the model under consideration has many first and second order numerical techniques in it, and if those techniques were replaced by second and fourth order techniques, the associated arithmetic requirements and memory capacity would be substantially reduced. Once again, new algorithms must be developed.

Because of the continuing demand for increased computational ability, future increases in supercomputer performance will be used to improve models rather than to reduce computational time and cost. Because scalar computers are approaching their maximal performance level, we may have to rely upon parallel computation to achieve the performance levels desired. Because of recent trends in supercomputer architecture, we already have significant experience with parallel computation in the form of vector processing and this experience has shown that parallel computation is making an impact on software and algorithms. Thus during the next decade we will see an increasing effort within DOE to develop methodologies for exploiting parallel computation.

REFERENCES

1. E. Yourdon, "How to be a Superprogrammer," Infosystems, February 1976, pp. 32-33.

2. J. Dongarra, "Some LINPACK Timings on the CRAY-1," Los Alamos Scientific Laboratory report LA-7389-MS (June 1978).

3. K. Fong and T. Jordan, "Some Linear Algebraic Algorithms and Their Performance on the CRAY-1," Los Alamos Scientific Laboratory report LA-6774 (June 1977).

4. P. F. Dubois and J. R. Kohn, "Equation of State Table Look-up: A Case Study in Vectorization," Lawrence Livermore Laboratory preprint UCRL-81184 (May 1978).