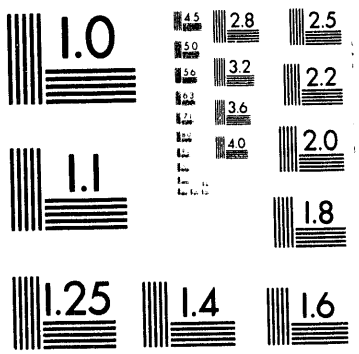
1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6

1 of 1

To be presented at the 1994 SCS Simulation Multiconference
April 11-13, 1994, San Diego, California

Parallel Monte Carlo Reactor Neutronics

by

Roger N. Blomquist and Forrest B. Brown
Reactor Analysis Division
Argonne National Laboratory
9700 South Cass Avenue
Argonne, IL 60439

* Work supported by the U.S. Department of Energy, Nuclear Energy Programs under Contract W-31-109-ENG-38.

PARALLEL MONTE CARLO REACTOR NEUTRONICS

Roger N. Blomquist and Forrest B. Brown
Reactor Analysis Division
Argonne National Laboratory
Argonne, IL 60439-4842

ABSTRACT

The issues affecting implementation of parallel algorithms for large-scale engineering Monte Carlo neutron transport simulations are discussed. For nuclear reactor calculations, these include load balancing, recoding effort, reproducibility, domain decomposition techniques, I/O minimization, and strategies for different parallel architectures. Two codes were parallelized and tested for performance. The architectures employed include SIMD, MIMD-distributed memory, and workstation network with uneven interactive load. Speedups linear with the number of nodes were achieved.

INTRODUCTION

The Monte Carlo method has been used for over 40 years to solve radiation transport problems in high energy physics, nuclear reactor analysis, radiation shielding, medical imaging, nuclear weapons design, etc. The wide applicability and extensive use of Monte Carlo is due to the generality and accuracy of the method — individual particle histories are simulated using random numbers, highly accurate representations of particle interaction probabilities, and exact models of 3-D problem geometry. The distribution of materials in the reactor is represented precisely by a set of common geometrical shapes, e.g., circular cylinders, so the absence of a computational mesh obviates the usual accompanying differencing approximations. The neutron reaction cross sections reflect the composition of each location in the reactor, and contain essentially no approximations in their energy dependence. Monte Carlo is sometimes the only viable method for analyzing complex, demanding particle transport problems, where one or more of these approximations is unwarranted.

Monte Carlo particle transport simulation of a nuclear reactor begins with a guess of the initial fission distribution, and performs an unaccelerated power iteration on the fission source, generation by generation. All the neutrons in a generation are tracked through the geometry from their birth to their absorption or leakage. The code must detect each boundary crossing and collision, and simulate the physical interaction of each collision. The reaction rate and flux scores are accumulated, and the fission sites spawned during collisions in fissionable media are stored for the next generation. Once all neutrons in a generation have been tracked, the fission sites produced by that generation are sampled to produce a set of neutrons which comprise the next generation. Therefore, only the histories within a generation are independent of each other; one generation's neutron population emerges from the previous generation's fission sites. The process is repeated, usually for many generations, until the desired precision has been obtained.

The principal limitation on the Monte Carlo method is computer power. Frequently, an average history involves 50 or more collisions, and hundreds of material interface crossings. Each generation may consist of over 10^4 histories, and hundreds of generations may be required to achieve results with acceptably low statistical uncertainty. It may be necessary to simulate millions of particle histories with billions of events, consuming many hours of supercomputer time. In the past few years, however, Monte Carlo methods have been successfully adapted to both SIMD and MIMD supercomputers, enabling the solution of "Grand Challenge" types of radiation transport problems in acceptable computing time.

Monte Carlo particle transport methods are naturally suited to parallel computers due to the inherent parallelism in the fundamental algorithm. Particle histories may be simulated independently and concurrently on separate processors, thus reducing the overall solution time in proportion to the number of processors. The independent history partitioning has been proven successful in tests performed on a wide variety of parallel computers. In practice, however, only a few existing Monte Carlo codes currently utilize parallel processing on a routine basis, for reasons discussed below.

The most significant recent computational advance is the migration of Monte Carlo calculations from expensive supercomputers to clusters of inexpensive workstations. Monte Carlo methods are ideal for single workstations or parallel processing on loosely-coupled clusters. The availability and power of distributed computing makes Monte Carlo more accessible, improving the quality of nuclear engineering designs.

GENERAL CONSIDERATIONS

SIMD Parallelism

Particle transport Monte Carlo was first adapted to SIMD (vector) parallelism in 1980 (Brown, 1981). The computational algorithm must be changed from the more traditional history-based scheme to an event-based scheme described below, which requires total restructuring of all data and extensive recoding. This has inhibited widespread application of so-called "vector Monte Carlo," despite very large potential gains. However, experience has shown that when the investment of vectorization is made, gains of 10x or greater in computational speed can be achieved.

MIMD and Distributed Parallelism

The key requirement for successful MIMD-distributed parallelization is sufficient memory, e.g., 16-32 MB, on each processor. This is the case either on true distributed-memory architectures, or on shared-memory processors where the memory is partitioned to mimic distributed memory. Then the Monte Carlo code and data can be replicated without difficulty, and each processor can independently follow a portion of the particles. Interprocessor communication is minimal, involving primarily the parcelling of histories to processors, and then the combining of results at the completion of all histories in a generation, using message-passing packages such as *PVM* (Beguelin, 1991) or *p4* (Butler, 1992). Existing Monte Carlo codes require only minor changes in a few high-level locations. Small Monte Carlo codes have been tested on a wide variety of MIMD parallel computers, and a few large production codes have been adapted to MIMD supercomputers such as the Cray-YMP (limited to only 8 processors). In all cases, gains in computational power were nearly proportional to the number of processors.

Domain Decomposition Issues

The most common and straightforward approach to parallel Monte Carlo is to partition the particles among processors, made possible because of the statistical independence of the particle histories. Even in some applications where the particle histories are not independent, such as where the particles might interact with the background medium and change its properties (e.g., isotopic number density or temperature), the histories can still be treated independently within sufficiently small time steps. Therefore, if one follows the physics and uses a "natural" partitioning of particle histories to processors, the inherent parallel nature of the physical process will be manifested in the algorithm. On the other hand, this mandates that each processor store in local memory the entire geometry description and properties of the media through which the particles are moving.

For large problems, domain decomposition by spatial region or by particle energy can be used to reduce the demand for memory by requiring that each processor store only part of the data. The first scheme assigns specific regions to processors, which only treat particles which are in their assigned regions. In the second, specific energy ranges are assigned to processors, which treat particles only in their energy ranges. There are two problems associated with both of these approaches: load-balancing and communications. Load-balancing is difficult without some a priori knowledge of the problem because particles may tend to congregate in certain regions of phase space, so each processor may end up working on a substantially different number of histories. Communication between processors increases because particles will cross boundaries or change energy in collisions, so that the particle description (10-20 words) must be communicated from the previous processor to the next. Thus, although domain decomposition may result in a substantial savings in memory requirement, it may suffer from increased interprocessor communication and unequal workloads.

Reproducibility of Results

Eigenvalue calculations are performed by sampling source sites to begin a generation, and then performing a random walk for each of the neutrons. Each slave process maintains it's own fission site bank, and at the conclusion of a generation these banks are concatenated into a master fission site bank on the master processor. To insure reproducibility, special techniques must be used (Brown, 1992) for both the source sampling and the random walk.

For each neutron in a generation, an initial random number seed is generated based on a specified "stride" through the random sequence. During the random walk, the random number generator is used to sequentially compute random numbers for a neutron based on its own initial seed. Each history is thus independent of all others, and hence the scores due to a neutron are independent of the order in which the

neutron histories are analyzed. This is sufficient to insure reproducibility within a generation.

Complete reproducibility requires a repeatable initial ordering of neutrons for each generation. This is assured by ordering the master fission bank in a unique and repeatable way using the following scheme: 1) at the start of a generation, assign each neutron a unique parent number and initialize a progeny counter for that neutron to zero; 2) when, during the random walk, a source site is to be added to the fission bank, increment the progeny counter and store its value and parent number in the bank along with the coordinates of the source site; 3) at the conclusion of the random walk, reorder the sites in the concatenated fission bank according to parent number and progeny counter values. After reordering the bank, the sampling of source sites for the next generation can proceed in the usual manner.

METHODS AND RESULTS

We describe here our experience with two particular Monte Carlo codes: Knolls Atomic Power Laboratory's *RACER* (Brown, 1986) and Argonne National Laboratory's *VIM* (Blomquist, 1991). Both are production-level codes used extensively for the engineering analysis of nuclear reactors. Each has been subjected to numerous quality-assurance checks and repeatedly verified through the analysis of benchmark experiments and comparisons with other codes. Both include continuous-energy collision physics and cross-section data. Both codes have been used extensively for reactor core analysis (eigenvalue problems) and reactor shielding analysis (fixed-source problems). Both will handle fully-detailed 3-D general geometry and fully-detailed lattice geometries — in principle, the geometry model will be "exact" if the user has enough patience in preparing the problem input.

Parallel/Vector Computing with *RACER*

RACER was developed at Knolls Atomic Power Laboratory in the 1980's for neutron transport (only). Consisting of about 20,000 lines of code, *RACER* was originally written for the Cyber-205 vector computer, a SIMD machine, and was entirely vectorized using CDC vector syntax and "q8-calls", all of which was later converted to standard Fortran-77 DO-loops for the Cray vector machines. In the late 1980's, *RACER* was adapted to MIMD parallelism on a Cray-YMP/8 and a Meiko Computing Surface-1 through the addition of about 200 lines of calls to message-passing routines using the *P3Lib* package (Fiedler, 1991). The resulting code is MIMD parallel at a high level, and SIMD parallel at a low level (on each processor), resulting in excellent performance on the Meiko and Cray-YMP systems, and extraordinarily high performance on the Cray-C90 computer.

RACER uses an event-based algorithm common to most vectorized Monte Carlo codes (Martin, 1987). In the original vectorized code, the neutron attributes are stored in one large stack. For each event type there is a queue containing pointers into the stack to those neutrons for which an event of that type is pending. During the calculation the pointers are moved from event queue to event queue based on the sequence of events. During the tracking process, the longest queue is selected and its pointers are used to gather the neutron attributes into a vector. The corresponding event is then analyzed, the altered neutron attributes scattered back into the stack, and the pointers moved to the appropriate queue for the next event.

In the new parallel version, the master provides each slave process a fraction of the neutron stack to work on. Each slave maintains its own event queues, and thus tracks its share of the neutrons independently of the other slaves. Synchronization is only necessary between neutron generations or when reaction rate tallies need to be passed to the master for combination with the tallies from the other slaves.

In the original vectorized version of the code, all of the microscopic and macroscopic cross section data, as well as derived quantities such as the isotopic scattering probabilities, were precomputed at every energy point and stored on disk during initialization. During the random walk, a portion of the data covering a fraction of the total energy range (a "supergroup") was read into memory and neutrons tracked until none were left within that supergroup. At that point, another supergroup of data was read in from disk and the process repeated. (In actual practice, the random walk calculations for one supergroup and the I/O for the next were overlapped, so that when the code finished the random walk for a supergroup it could immediately begin the next.)

Since the *RACER* parallelization is over neutron histories — as opposed to a spatial or other decomposition — all slave processes need access to the entire set of cross section data for each of thousands of materials. The macroscopic cross section data management scheme used in the sequential code would lead to a problem for most reactor calculations on massively parallel machines; the limited memory per node would permit only a small fraction of the macroscopic data to be in memory at a time. Furthermore, disk I/O would not scale as fast as the computation with increasing numbers of slaves. To overcome these difficulties, the parallelized *RACER* recomputes macroscopic quantities as needed rather than precomputing all of the data. Since the added

computations scale with the number of processors, and the I/O does not, the overall code performance became scalable.

A comparison of *RACER* performance on three different parallel computers — the Cray-C90, Cray-YMP, and Meiko-CS1 — is given in Figure 1 (Sutton, 1994). It should be noted that the performance on a single i860 processor of the Meiko system is roughly comparable (within a factor of about 5) to that of *RACER* or *VIM* on a single Sun sparc2 or IBM rs6000. The overall range of performance data, from one i860 processor to 16 processors on the Cray-C90, spans a range of 1,000x in relative performance.

Parallel Distributed Computing with *VIM*

VIM has been developed and supported by Argonne since the late 1960's. The code is used for either neutron or gamma-ray transport, and can also handle multigroup cross-section datasets, as well as continuous energy data. It consists of about 30,000 lines of machine-portable Fortran-77. Due to the vintage of its syntax, the code is strictly scalar, and no attempt has been made at vectorization. *VIM* has been used on a very large number of different computers, including Sun-sparc2, IBM-rs6000, Cray, IBM-3084, VAX, CDC, etc. Recently, *VIM* has been adapted to distributed parallel processing on a workstation network using the *p4* message-passing package.

Most applications of *VIM* are run using Sun workstations on the Reactor Analysis Division computing network at Argonne. *VIM* is the most computationally-intensive task run on the network; a single job may run continuously for days or weeks on one workstation. The Distributed Queuing System, *DQS*, (Green, 1992) assigns these and thousands of shorter batch jobs each month to idle or lightly loaded nodes. Current *VIM* parallel processing development uses the *p4* message software package developed at Argonne, with *DQS* allocating a "virtual supercomputer", i.e., several machines, to a parallel *VIM* job. Parallel *VIM* calculations are intended to be scalable to any number and combination of Sun and IBM workstations in the network. In addition, the parallel code could be run on Argonne's new parallel computer, the IBM SP1, without changes to the source code.

The parallel version of *VIM* was developed in a manner similar to the *RACER* parallel algorithm. A "master/slave" approach is used, with particle histories partitioned among the slave processes. However, since *VIM* is run primarily on a workstation network, the processing speed of each node can differ markedly from the others. First, the individual workstations may have different processor types and speeds. Second, the interactive and system loads on each machine will vary in an unpredictable manner, even during a calculation. Third, it is even possible for a processor to get so busy that it becomes effectively unavailable. In effect, the "virtual supercomputer" configuration and performance could change continuously during the course of a single calculation. To handle these difficulties, the histories within a given generation are not statically partitioned among processors. Instead, they are processed in "chunks" of a few hundred or thousand histories at a time, sent only to those slave processes which ask for work. Whenever a slave process is ready to accept new work, it signals the master, which then sends a "chunk" of work to the waiting slave. In theory, one machine could wind up handling all of the histories. In practice, however, the chunks of work for long calculations have been found to be distributed among processors in rough proportion to their relative CPU speeds.

Parallel *VIM* calculations have achieved reductions in computing time in direct proportion to the number of distributed workstations utilized, even on a moderately busy network. As shown in Figure 2, speedups have been nearly linear with the number of workstations, as long as not all of the available machines were used. That is, during the course of a typical calculation, there are always several background network activities (e.g., file backup, mail processing, NFS file transfers, etc.) which may execute at high priority and monopolize a particular machine's CPU cycles. As long as *DQS* is asked to allocate some, but not all, of the machines to a calculation, it will not allocate the machines which are heavily loaded with system tasks to the *VIM* calculation. Using the "virtual supercomputer" provided by a workstation network, a typical Monte Carlo calculation which previously required several days or weeks of computing time on a single workstation can now be completed overnight using 4-8 machines.

CONCLUSIONS

The conclusion from this experience is that a message-passing programming paradigm provides excellent portability across different MIMD machines (i.e., those supported by the message-passing subroutine libraries) and reduced code development costs (since nearly all original coding can be reused). For conventional scalar Monte Carlo codes, a MIMD, message-passing approach is recommended over the more machine-specific and labor-intensive SIMD approach. For codes which have already been adapted to SIMD parallelism, the transition to a hierarchical parallelism (high-levels MIMD, low-level SIMD) is straightforward, and the multiple levels of parallelism can provide significant performance benefits.

With the possibility of simultaneously using thousands of processors, the methods described here must be enhanced to accommodate a changing system configuration. Fault-tolerant algorithms should alter the mapping of Monte Carlo processes to physical nodes if one node is too slow or fails to respond in reasonable time. Practical schemes for monitoring the number of nodes, relative speeds, network communication speeds, etc., should also be developed and incorporated into the higher-level Monte Carlo algorithm. Dynamic load balancing should distribute work automatically across system resources in a manner which minimizes the overall computing time, subject to constraints such as minimal interference with interactive computing processes. Such considerations are new to scientific computing.

The potential payoffs from exploiting parallelism in particle transport Monte Carlo are far reaching. History suggests that gains of only 2-3 in computer power result in more calculations and incremental improvements in engineering, while breakthroughs in innovative computational and design methods require much larger gains. The application of Monte Carlo to nuclear reactor depletion, for example, is feasible now only on very large supercomputers, and even then only for selected portions of the reactor. Gains of 100-1000 in computational power due to parallelism would permit fully detailed, 3-D modeling of depletion for entire reactors. For more modest calculations, smaller speedups would permit the use of Monte Carlo interactively (i.e., results in seconds, not hours or days), greatly enhancing the productivity of design engineers. The potential payoffs, when coupled with the cost-effectiveness of parallel computers, are very large.

ACKNOWLEDGEMENTS

The authors are indebted to a number of coworkers who contributed directly or indirectly to the work described herein. Particular thanks are due to T. M. Sutton (KAPL), W. R. Martin (Univ. of Michigan), J. A. Rathkopf (LLNL), and D. M. Malon(ANL).

REFERENCES

Beguelin, A., et. al. 1991. "A User's Guide to PVM -- Parallel Virtual Machine", *ORNL/TM-11826*, Oak Ridge National Laboratory.

Blomquist, R. N. 1991. "VIM." *Proceedings of the International Topical. Mtg Advances in Mathematics, Computations, and Reactor Physics* (Pittsburgh, PA, April 28 - May 2). American Nuclear Society.

Brown, F. B., W. R. Martin, and D. A. Calahan. 1981. "Investigation of Vectorized Monte Carlo Algorithms." *Transactions of the American Nuclear Society*, no. **39**: 755.

Brown, F. B. 1986. "Vectorization of 3-D General Geometry Monte Carlo." *Transactions of the American Nuclear Society*, no. **53**: 283.

Brown, F. B. and T. M. Sutton. 1992. "Reproducibility and Monte Carlo Eigenvalue Calculations", *Transactions of the American Nuclear Society, no.* **65**: 234.

Butler, R. and E. Lusk. 1992. "User's Guide to the p4 Programming System." ANL-92/17. Argonne National Laboratory.

Fiedler, S. L. 1991. "Portable Parallel Programming," KAPL-4728. Knolls Atomic Power Laboratory. Schenectady, NY. (April)

Green, T. P. 1992. "The Distributed Queuing System." Supercomputer Computations Research Institute, Florida State University.

Martin, W. R. and F. B. Brown. 1987. "Present Status of Vectorized Monte Carlo for Particle Transport Analysis." *International Journal of Supercomputer Applications* no. **1**: 11-32.

Sutton, T. M. and F. B. Brown. 1994. "Parallel Monte Carlo for Reactor Calculations." *Proceedings of the International Topical Meeting on Advances in Reactor Physics* (Knoxville, TN, April 11-15). American Nuclear Society.

THE AUTHORS

Roger Blomquist has been a nuclear engineer at ANL since 1979, primarily developing Monte Carlo neutral particle transport methods. He has also developed computational thermal-hydraulics methods, and performed the nuclear design for the Intense Pulsed Neutron Source. His degrees are: B.S., Physics, The College of William and Mary; M.S. and Ph.D., Nuclear Engineering, Northwestern University.

Forrest Brown was the lead physicist for Monte Carlo Development at KAPL from 1981-88, and manager of Reactor Physics Computations during 1988-92. He also benchmarked many advanced computer systems. His degrees are: B. S. and M.S., Physics, Rensselaer Polytechnic Institute; Ph.D., Nuclear Engineering, University of Michigan.

Both authors served in the U.S. Navy's nuclear propulsion program.

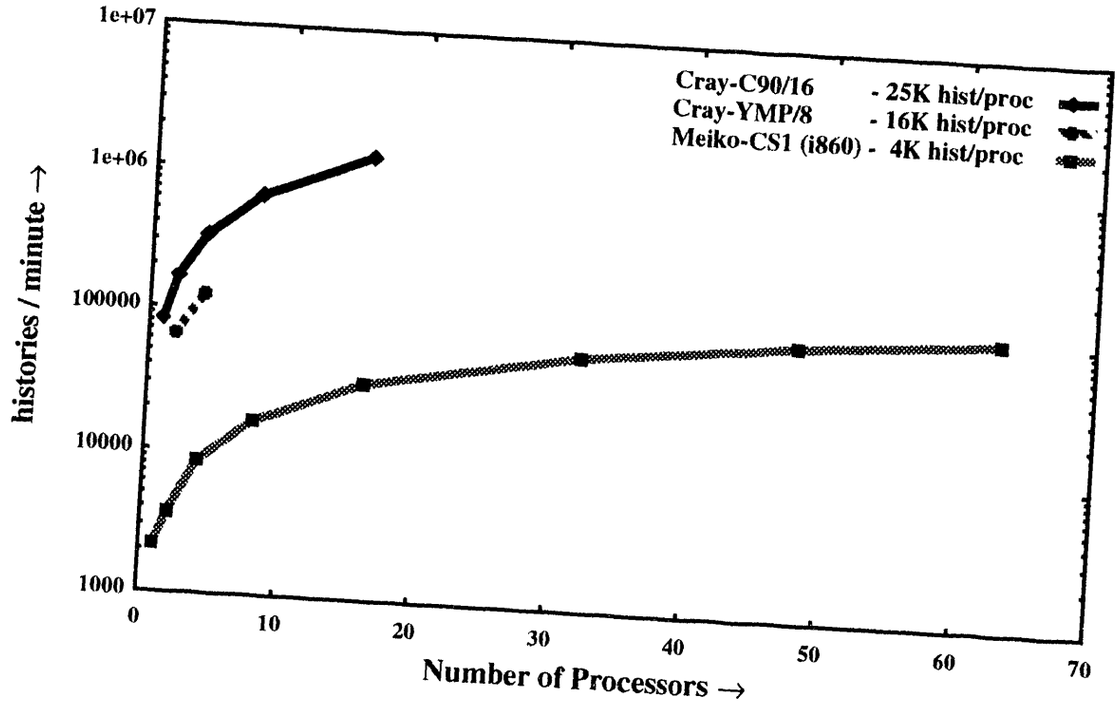

Figure 1. Parallel /Vector *RACER* Performance

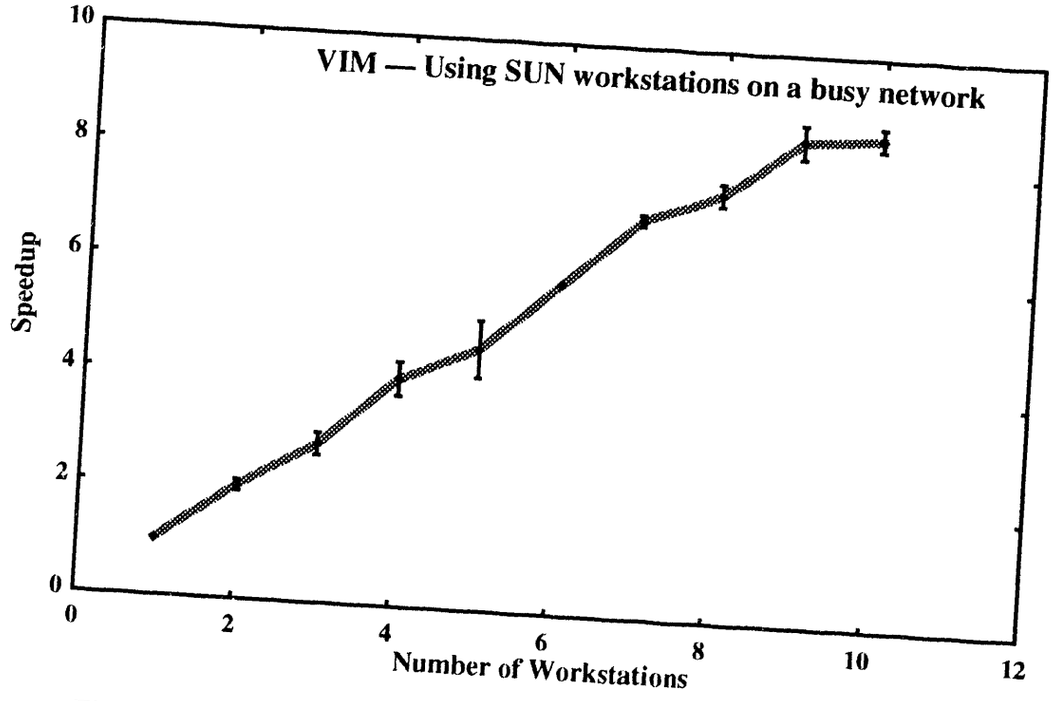

Figure 2. VIM Performance with Distributed Parallel Processing

DATE FILMED

4 / 12 / 94

END