Paper for Invited Presentation

Prepared for the
1994 ANS Topical Meeting on Advances in Reactor Physics
(April 11-15, 1994, Knoxville TN)

Session: ***Algorithms for New Computer Architectures***
Organizer: D. J. Kropaczek (NCSU)

Distributed Computing and Nuclear Reactor Analysis*

Forrest B. Brown, Keith L. Derstine, Roger N. Blomquist

Argonne National Laboratory
Reactor Analysis Division
9700 S. Cass Ave. — RA/208
Argonne IL 60439-4842

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

DISTRIBUTED COMPUTING AND NUCLEAR REACTOR ANALYSIS

Forrest B. Brown
Keith L. Derstine
Roger N. Blomquist

Argonne National Laboratory, 9700 S. Cass Ave., Argonne IL 60439-4842, (708) 252-9622

KEYWORDS: computation, parallel computing, workstations

ABSTRACT

Large-scale scientific and engineering calculations for nuclear reactor analysis can now be carried out effectively in a distributed computing environment, at costs far lower than for traditional mainframes. The distributed computing environment must include support for traditional system services, such as a queuing system for batch work, reliable filesystem backups, and parallel processing capabilities for large jobs. All ANL computer codes for reactor analysis have been adapted successfully to a distributed system based on workstations and X-terminals. Distributed parallel processing has been demonstrated to be effective for long-running Monte Carlo calculations.

INTRODUCTION

Large-scale scientific and engineering calculations can now be carried out by three different general approaches: *traditional computing*, where all number-crunching is performed on powerful centralized computers or supercomputers; *MP computing*, where the number-crunching is performed on a massively parallel system with hundreds or thousands of tightly-coupled identical processors; and *distributed computing*, where a geographically dispersed, possibly heterogeneous collection of separate computers is used. Traditional computing methodology is mature, with a rich collection of well-known algorithms and computer codes for solving scientific problems. After nearly two decades of experience with CRAY computers, vectorization and modestly parallel algorithms (~16 or fewer processors) are well-understood and used effectively. Massive parallelism is relatively new, and offers dramatic improvements in cost/performance for certain classes of problems. Considerable effort in code and algorithm development is often required to realize this potential for many engineering applications, however. The third approach, distributed computing, is also relatively new, but has perhaps the greatest potential for radically improving the state of scientific computing.

In recent years there has been an explosive growth in the power and availability of desktop, Unix-based, engineering workstations. These machines were cost-justified and purchased to improve the productivity of engineers, and are typically connected on local-area-networks within a division or company. The widespread availability of workstations has enabled a revolution in computing: Because the combined power of 5-10 workstations rivals that of a supercomputer, supercomputer power is now available to anyone — at no additional hardware cost. Distributed computing provides a "virtual supercomputer" to anyone having the proper software to coordinate and control a distributed parallel calculation. As shown in Figure 1, the type of computer, processing power, and unique capabilities of various computers can be selectively matched to particular applications, in order to create the most effective "virtual supercomputer"

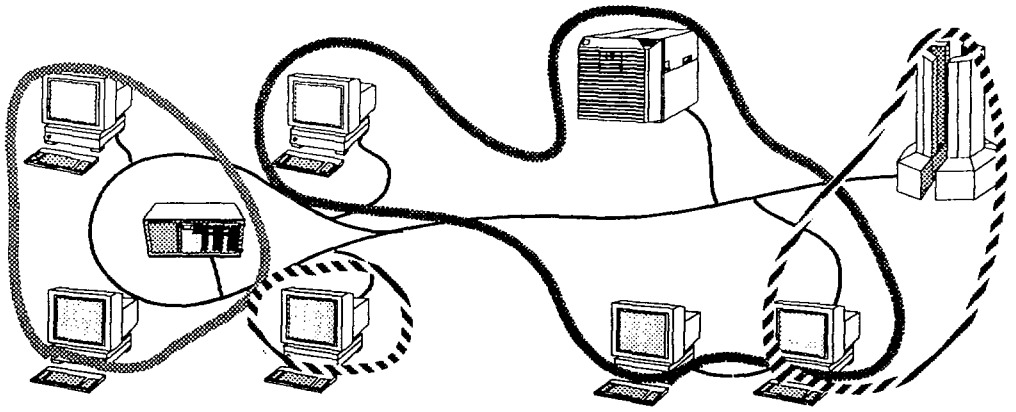

Figure 1. Virtual Supercomputer Example

for a particular calculation. This can be particularly effective at night, when workstations are typically unused. As a result, we can today talk about 3 classes of supercomputing for large-scale scientific applications: "real" supercomputing on traditional computers (e.g., CRAY), "cheap" supercomputing on massively parallel computers, and "free" supercomputing using distributed computing on a workstation network.

REQUIREMENTS FOR EFFECTIVE DISTRIBUTED COMPUTING

For distributed computing to be effective and practical for nuclear reactor system analysis, it is not enough to merely provide powerful workstations to the engineers and scientists. New distributed capabilities and system-wide supporting features must be made available, just as in traditional computing. To perform the large-scale computations required for nuclear reactor system analysis, a number of requirements must be met: (1) A job queuing system must be available which can accommodate any combination of "large numbers of short jobs" and "small numbers of very large or long-running jobs" which are submitted to the distributed system in a batch mode. This system must automatically schedule these jobs according to various site priorities, locate a free or lightly loaded machine, route a job to that machine, handle job cleanup and output dispersal, etc. While many such systems are available on traditional mainframes, *distributed* queuing systems are relatively new. In addition to balancing the load across various networked machines, maximizing throughput for the entire system, and dealing with a heterogeneous collection of machines, there is the additional complication of a dynamic computer configuration, as for example, when various workstations are turned on or off. (2) To accommodate very large computational tasks, modestly parallel processing must be supported in the distributed system. A distributed system can be viewed as just another form of MIMD computer, albeit with very slow communication between nodes. A message-passing approach to parallel computation using standard packages (e.g., PVM[1], P4[2]) is relatively straightforward. The most interesting challenges for algorithm development involve the heterogeneous, possibly time-dependent nature of a distributed system — load-balancing and fault-tolerance become crucial for codes to be robust. (3) Users, their work, and applications software must be adapted to a distributed environment. Many "standard" nuclear engineering codes were developed for mainframes and must be modified to work properly and accurately on 32-bit Unix workstations. Some workstation compilers do not tolerate all of the ancient Fortran-66 (or pre-Fortran-66) constructs found in many codes. Specialized routines such as random number generators must frequently be rewritten or redesigned, and former assembly-

language routines must be converted to C. For most applications, Unix shell scripts must be developed to replace the various JCL procedures or "exec's" used to run the codes on traditional mainframes. For significant advances in the productivity of individual engineers and scientists, it is desirable to develop graphical user interfaces (GUI's) to simplify input for complex models and to provide visualization tools to enhance the understanding of results.

EXPERIENCE WITH DISTRIBUTED COMPUTING & REACTOR ANALYSIS

For over 30 years, Argonne National Laboratory has developed and applied large-scale software systems for the engineering analysis of nuclear reactor plants. Large code systems such as VIM (3D continuous energy Monte Carlo), DIF3D (3D finite-difference or nodal diffusion theory), MCC-2 (cross-section generation), REBUS3 (reactor core burnup), SASSYS (transient analysis of reactor plants), etc., were developed and run on traditional mainframe computers and supercomputers. In the past two years, all of the reactor analysis codes and supporting software have been adapted to a distributed computing environment, the "RA Network." In addition, a large number of standard, externally-developed code packages have been made available, including CASMO/SIMULATE, MCNP, TWODANT, MATHEMATICA, MATLAB, ORACLE, SAS, DISSPLA, etc. An integrated support environment and distributed computing methodology were established on the RA Network, including the following major components: (1) Unix operating systems, with tcp/ip networking, (2) special configuration of the memory, disk, and scratch space on each workstation to accommodate both interactive and batch usage for large jobs, (3) DQS[3], the Distributed Queueing System from Florida State University, to manage the scheduling and load balancing for batch jobs, (4) the PVM software package from Oak Ridge National Laboratory to provide a framework for parallel processing of large applications, (5) the P4 package from Argonne to provide additional support for parallel applications, (6) numerous Bourne shell, C shell, and *perl* procedures to provide transition and convenience features for a diverse user group, (7) libraries of compatibility routines for I/O and environmental support, (8) source code management and version control using SCCS, (9) frequent and reliable backup services, and (10) continued development of the large production codes to improve their performance in a distributed computing environment.

At present, the Division workload is fully supported by a network of 15 SUN workstations, 2 IBM RS6000 workstations, and 60 NCD X-terminals, as illustrated in Figure 2. The RA Network is somewhat atypical in its heavy reliance on X-terminals. We have found X-terminals, however, to be an extremely cost-effective approach for nuclear engineering work. Nearly all users have a need to occasionally run very large calculations, where 32 MB of memory and/or 1GB of disk storage or more are needed. Sharing of these resources via X-terminals is very inexpensive compared to purchasing a fully-configured workstation for each user. In addition, the X-terminals are well-received due to their small footprint, silent operation, and fast response time. Locating all workstations in 2 centralized, lockable areas also enhances system security without placing undue restrictions on users. This configuration of remote X-terminals and geographically concentrated workstations also simplifies system administration tasks. Standard remote configuration files, X-server code and *xdm* server daemons are installed on appropriate workstation hosts and shared by X-terminals. Additional X-terminals may then be added to the network in a matter of minutes. Separating the display technology from the computational resources makes it feasible to expand the network resources in an evolutionary manner.

In order to accommodate both batch and interactive work effectively, special attention must be given to the configuration of each workstation in the network. Each workstation is required to have at least 64 MB of memory in order to permit simultaneous use by several interactive users and at least one batch job, without resorting to excessive swapping/paging to disk. A corresponding swap area of at least 3 x 64 MB must be provided and (on SUN's) shared with */tmp* via the "TMPFS" configuration option. In addition, each workstation is required to have at least 1 GB of local disk for scratch files, so that scratch disk usage

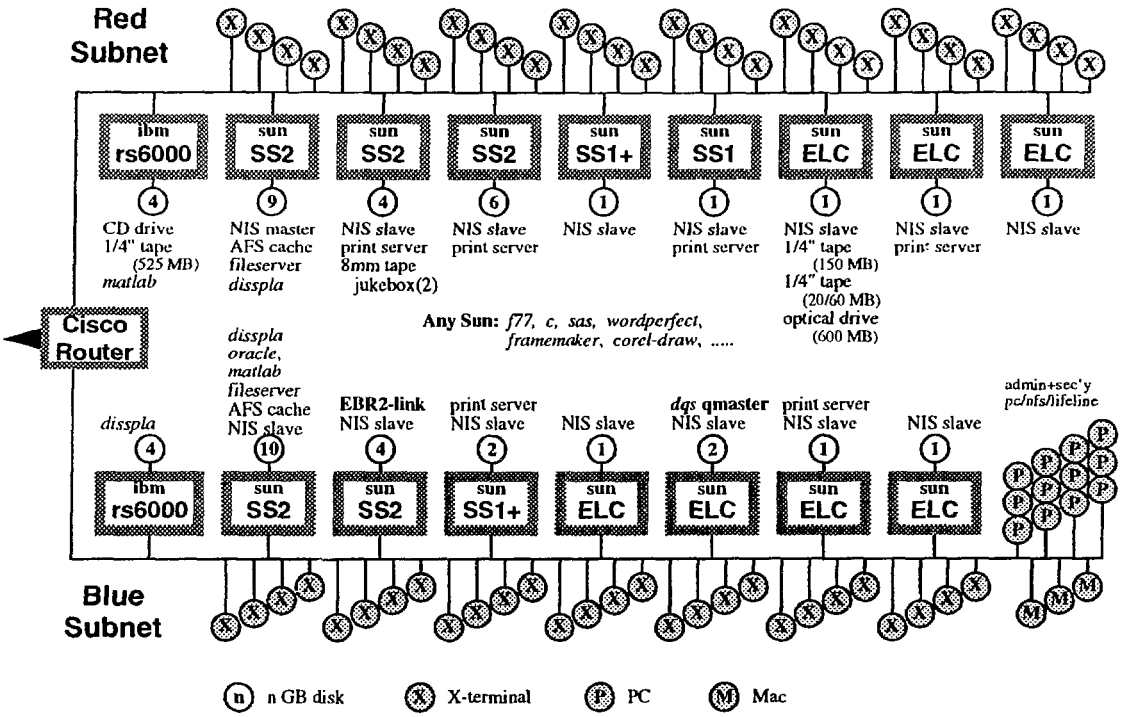

Figure 2. Reactor Analysis Network

(in a local */var/tmp* filesystem) does not impact the overall network traffic. The separation of */tmp* and */var/tmp* filesystems provides interactive users with protection from batch jobs which use excessive amounts of */var/tmp* space. Machine-specific executables and libraries are stored in a consistent manner on all machines (e.g., */usr/local/bin*, */usr/local/lib*), and all *home* filesystems are NFS-automounted from every machine. Four GB of public file systems are also NFS-automounted to all hosts for short-term (< 30 days), emergency storage of large files. This single-image, machine-independent file system model supported by guaranteed hardware configurations permits batch jobs to run on any host.

A batch job computing strategy consistent with the workstation configuration described above imposes several conventions on the users: (1) Batch jobs should normally be run using a working directory located in a workstation's local scratch area (e.g., */var/tmp/$LOGNAME/...*); (2) Batch jobs should copy executables and I/O intensive data files to the local scratch area to minimize network traffic, and to avoid interference between long-running jobs and updates to executables; (3) All files in the scratch area must be deleted at the conclusion of a job to guarantee future scratch space availability. A collection of Bourne shell scripts (RASHELL) and *perl* scripts was created to implement this strategy. It is the user's responsibility to ensure that, prior to job termination, a job script retains interface files in directories other than the scratch area. The RASHELL scripts also consolidate and route job *stdout, stderr* and Fortran-formatted output to a user-specified directory and/or microfiche, laser printers, or line printers.

The Distributed Queuing System, DQS, was installed on the RA network and tailored to meet Division needs. All batch work on the RA Network is submitted to DQS, which locates the proper machine

resources, schedules the work, and initiates execution of batch jobs. DQS provides load-balancing in a distributed system via a "group" mechanism: Batch queues reside on individual workstations and handle requests on a FIFO basis. Several batch queues may be grouped together, and jobs submitted to that group can be scheduled on the next available, least-loaded machine in the group by DQS. For application to the RA Network, one "normal" job queue is assigned to each SUN workstation and 3 queues to each RS6000 workstation, with low execution-priority (*"nice 19"*) relative to interactive work. (For typical RA Division codes, an RS6000 was measured to be about 3-5 times faster than a SUN SPARC2. In addition, throughput was found to be highest for a mix of several CPU- and I/O-intensive jobs.) In addition, each workstation is configured with a "now" queue, which executes at somewhat higher priority (*"nice 12"*), has a time limit to prevent long-running jobs, and while active results in suspension of the "normal" queue on the same machine. All "normal" queues on SUN workstations are combined into a "sun" group, the "now" queues on SUN's into a "sun.now" group, and similarly for the "ibm" and "ibm.now" groups. The intent is to permit only 1 batch job to execute at a time on a given SUN (3 for RS6000's), with DQS scheduling the work in the most efficient manner among the various groups of workstations. This arrangement has functioned very well, handling about 3,000 batch jobs per month with little or no impact on the interactive response time.

To support parallel distributed processing on the RA Network, the P4 and PVM software packages are provided. The two packages are functionally similar, supplying libraries of Fortran- or C-callable subroutines which handle the startup of processes and communication between processes on (possibly) different machines. These packages may be used for a "message-passing" approach to loosely-coupled, parallel processing on a workstation network. In addition, codes developed in such a manner are very portable, and can generally be run with few or no changes on other machines supported by P4 or PVM. For parallel processing to be effective on a distributed system, there must be close cooperation between the message-passing system (P4 or PVM) and the job scheduler (DQS). DQS provides direct support for parallel jobs, in that its scheduling and resource allocation schemes specifically allow multiple workstations to be assigned to a specific job. Other batch work is subsequently scheduled on different machines, to avoid interfering with the parallel job.

PARALLEL DISTRIBUTED COMPUTING WITH VIM

Monte Carlo simulations provide a highly accurate, but computationally intensive, means of analyzing the physics behavior of nuclear reactors. The VIM[4] Monte Carlo code from Argonne is a well-known, mature, and proven code, with no essential approximations to the geometric modelling or collision physics. VIM can be used to provide "truth calculations," or computational benchmarks, which may be used to calibrate and fine-tune various faster-running, but less accurate, computational methods. The code is used for either neutron or gamma-ray transport, and can handle both continuous-energy and multigroup cross-section datasets. VIM has been used extensively for reactor core analysis (eigenvalue problems) and reactor shielding analysis (fixed-source problems), and will handle fully-detailed 3-D general geometry and lattice geometries. In principle, the geometry model will be "exact" if the user has enough patience in preparing the problem input.

VIM has been developed and supported by Argonne National Laboratory since the late 1960's. It consists of about 30,000 lines of machine-portable Fortran-77. Due to the age of the code and its relatively poor structure and coding 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. VIM Monte Carlo is the most computationally-intensive task run on the RA Network. A single VIM calculation may run continuously for days or weeks on a single workstation. The DQS batch system handles these jobs effectively, in addition to thousands of shorter jobs submitted each month. Current VIM development is focused on parallel processing in a distributed computing environment, using the P4 parallel processing software package developed at Argonne. The concept

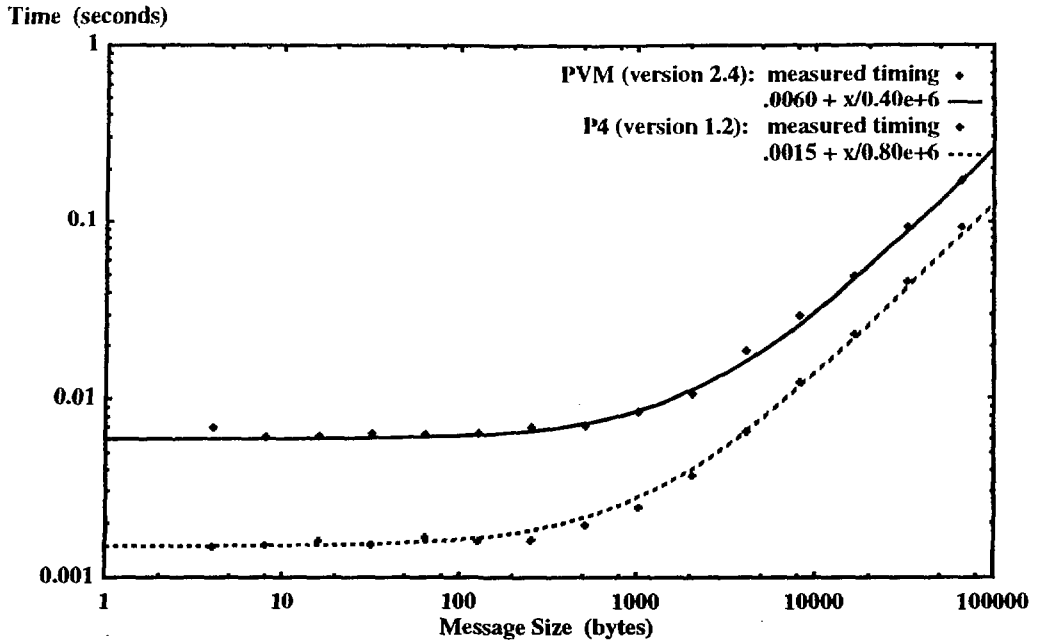

Figure 3. Timing for PVM & P4 Message Passing Using SUNs+Ethernet

of a "virtual supercomputer" is supported by DQS, so that several machines may be allocated to a single VIM job.

Monte Carlo particle transport methods are ideally suited to MIMD parallel computers due to the inherent parallelism in the fundamental algorithm. A distributed computing system provides a form of MIMD parallel computer, with slow, Ethernnet-based communications between nodes. Figure 3 shows timing measurements for one-way sending/receiving of messages of various lengths between two SUN workstations on a local Ethernet-based network, using the P4 and PVM message-passing packages. It can be seen that latencies are large, in the 1-10 millisec range, and that data streaming rates are somewhat less than 1 MB/sec. Message-passing for distributed parallel processing is thus roughly similar to disk I/O, and is many orders of magnitude slower than (local) memory access. While the relatively slow message-passing available on a distributed workstation network will inhibit many types of parallel calculation, parallel Monte Carlo can still be very effective since interprocessor communication is minimal, involving primarily the combining of results at the completion of all histories. If sufficient memory is available to each processor, the Monte Carlo code and data can be replicated, and each processor can independently follow a portion of the particles. Existing Monte Carlo codes require changes in only a few high-level locations. Unlike vectorization, where nearly all coding must be rewritten, a MIMD or distributed parallel approach to Monte Carlo permits the resuse of nearly all coding without change. The key requirement for widespread applicability of particle transport Monte Carlo on MIMD or distributed computers is memory size. At least 16 MB per processor is required to hold a complete copy of the Monte Carlo code and data for most applications, and many large problems (for which parallelism is crucial) require 32 MB or more per processor.

Parallel VIM calculations are intended to be scalable to any number and combination of SUN and IBM

workstations in the network. A "master/slave" approach is used, with particle histories (in the current generation) partitioned among the slave processes. Each slave 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 process for combination with the tallies from the other slaves. Special consideration is given to the manner in which work is partitioned among processors: Since VIM is run primarily on a workstation network, there is a large potential for wide disparities in the processing speed of each node. First, the individual workstations used in a parallel calculation 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 particle histories within a given batch (generation) are not statically partitioned among processors. The histories 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. The master process then sends a "chunk" of work to the ready 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.

To insure reproducibility, special techniques must be used for both the the random walk and the source sampling used in eigenvalue calculations.[5] 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 neutron 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. During a generation of neutron histories, induced fission sites are stored in a "fission bank" for later use in sampling the source for the next generation. Each slave process maintains it's own fission bank, and at the conclusion of a generation the individual banks are concatenated into a master bank on the master process. A unique reordering of the progeny in the fission bank, according to their parent neutron number, is accomplished using the algorithm of Ref. 5.

Parallel VIM calculations have achieved reductions in computing time in direct proportion to the number of distributed workstations utilized. While parallel speedups vary somewhat depending on problem type, typical performance results using multiple SUN workstations on the RA Network are shown in Figure 4 for a VIM analysis of the TREAT reactor. Even on a moderately busy network, 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.

FUTURE NEEDS & WORK IN PROGRESS

The Reactor Analysis Division has fully embraced distributed computing and is now performing more calculations, at much lower cost, than ever before. Our experience in applying a distributed computing approach to nuclear reactor analysis has made us aware of a number of challenges and opportunities for continued development. To be effective on a variety of parallel and distributed computer systems, future

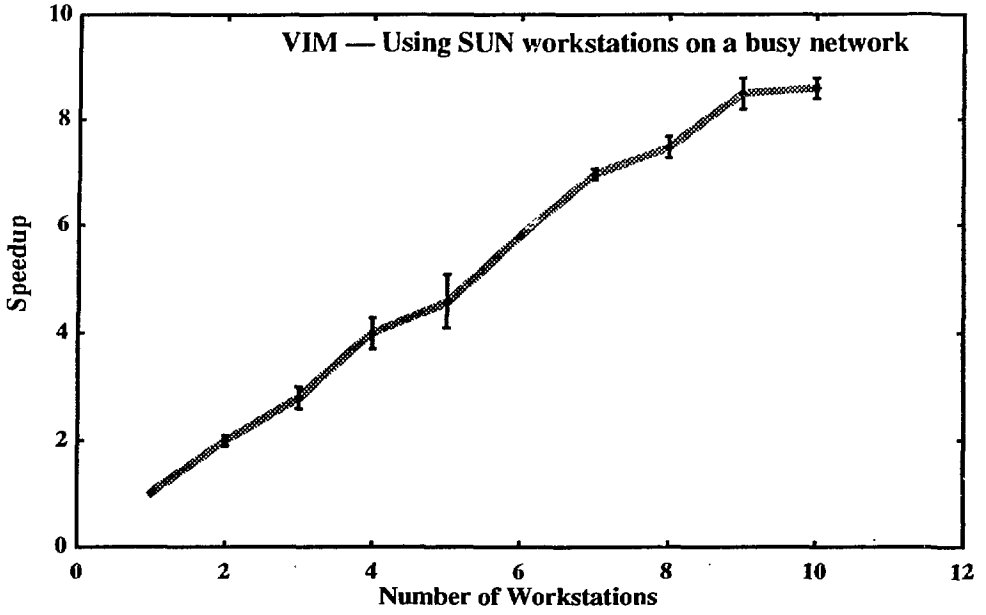

Figure 4. VIM Performance with Distributed Parallel Processing

large-scale scientific software must be able to adapt itself to changes in the computing environment. Key features are fault tolerance, dynamic load balancing, and multiple levels of parallelism in the coding and algorithms. Monte Carlo codes are generally early-adopters of advanced features, since the codes are adaptable to any known computer architecture. It is reasonable to expect that speedups of 100-1000 times that of current computers will be possible in the near future, allowing for unprecedented applications for Monte Carlo simulation. Success in adapting Monte Carlo methods will provide experience and expertise in robust, fault-tolerant scientific computation, paving the way for applications in other areas. It is our plan to investigate these areas first for Monte Carlo, and then to extend the lessons-learned to other large codes. Particular areas of interest are summarized below:

- Several production-level Monte Carlo codes for particle transport analysis are now available for distributed or parallel computer systems, including VIM (ANL), MCNP (LANL), and RACER (KAPL). These codes all employ a "master/slave" algorithm and cannot adapt dynamically to changes in the computer configuration. The loss of even one node results in termination of a calculation. While these codes are effective on current machines with a few dozen nodes (moderately parallel), their effective use on distributed or massively parallel computers with 1,000's of nodes requires a more robust approach. The master/slave approach should be discarded, opting instead for a hierarchy of parallel processes. Multiple levels of parallelism (e.g., clusters of nodes, and clusters of clusters) will reduce the amount of contention and wait-time for CPU and communications resources, and will provide flexibility for novel architectures.

- Software and algorithms for scientific computations must begin to incorporate fault tolerance, dynamic load balancing, and capabilities for reconfiguration. Fault-tolerant algorithms should be developed such that, if one node is too slow or fails to respond in reasonable time, the code will alter the mapping of processes to physical nodes, to adapt efficiently to the current environment. Practical

schemes for monitoring and querying the total computing environment (e.g., number of nodes, relative speeds, network communication speeds, etc.) should also be developed and incorporated into high-level computational algorithms. Dynamic load balancing should be developed such that work will be distributed automatically across system resources in a manner which minimizes the overall computing time, subject to constraints such as minimal interference with interactive computing processes.

• Transport codes based on collision probability methods are generally well-suited to distributed parallel computing, since the elements of the collision probability matrices can be determined independently, without significant interprocessor communication, and the resulting matrix equations are large enough to solve efficiently in parallel. The GTRAN2[6] general-geometry transport code has been successfully adapted to distributed parallel processing for the equation solution,[7] and development of parallel methods for matrix construction is in progress.[8] As for Monte Carlo codes, further development is required to implement robust, fault-tolerant algorithms with dynamic load balancing.

• Large-scale computer codes for diffusion theory analysis, diserete ordinates transport, fluid dynamics, heat transfer, and two-phase flow should be adapted to distributed parallel processing. These types of code can involve significantly greater amounts of interprocessor communications than Monte Carlo codes, making their performance sensitive to the particular parallel algorithm implemented and the computer architecture. Since these codes use finite-difference or finite-element methods and many mesh points, domain decomposition strategies should be developed which minimize the relative amount of communications.

• Some other codes may not be well-suited to distributed parallel processing. Plant safety analysis codes, for example, typically involve the processing of very many timesteps with relatively little computational work at each step which can be performed in parallel. Similarly, nodal diffusion theory codes may solve problems with very small numbers of nodes. Distributed parallel processing may not be practical for these codes since the slow network communications between machines could more than offset any gains from performing a small amount of computation in parallel. It appears at present that single calculations using these types of codes are effective only on fast uni-processors. When repeated calculations or parameter studies are required, however, job parallelism may be used along with DQS to exploit the full potential of a workstation network. Further research in these areas may be warranted as network communication speeds improve.

It is expected that the algorithms and computing techniques developed for these reactor analysis applications will be applicable to many other applications of interest to both the DOE and US industries.

REFERENCES

1. A. BEGUELIN, ET AL, "A Users' Guide to PVM — Parallel Virtual Machine," *ORNL/TM-11826*, Oak Ridge National Laboratory (1991).

2. R. BUTLER and E. LUSK, "User's Guide to the p4 Programming System," *ANL-92/17*, Argonne National Laboratory (1992).

3. T. P. GREEN, "The Distributed Queuing System," Supercomputer Computations Research Institute, Florida State University (1992).

4. R. N. BLOMQUIST, "VIM," *Proc. Int. Topl. Mtg. Advances in Mathematics, Computations, and Reactor Physics*, Pittsburgh, PA, April 28-May 2, 1991, American Nuclear Society (1991).

5. F. B. BROWN and T. M. SUTTON, "Reproducibility and Monte Carlo Eigenvalue Calculations," *Trans. Am. Nucl. Soc.* **65**, 234 (1992).

6. J. VUJIC, "GTRAN2 — A General Geometry Transport Theory Code in 2D," *Proc. Int. Topl. Mtg. Advances in Mathematics, Computations, and Reactor Physics*, Pittsburgh, PA, April 28-May 2, 1991, American Nuclear Society (1991).

7. W. J. WILSON, J. L. VUJIC, and A. G. GU, "Parallel Multiple-Assembly Calculations in GTRAN2/M," *Trans. Am. Nucl. Soc.* **69**, 204 (1993).

8. J. VUJIC, private communication (1993).