APPLICATIONS, ALGORITHMS, AND SOFTWARE FOR MASSIVELY PARALLEL COMPUTING

Robert E. Benner and Joe M. Harris

Robert E. Benner *and* ***Joe M. Harris*** *work for Sandia National Laboratories, Mr. Benner in Albuquerque, New Mexico, and Mr. Harris in Livermore, California. Mr. Benner is a senior member of technical staff in the Mathematics and Computational Science Department. He works on science, engineering, and defense applications for massively parallel computers and parallel numerical and computer science algorithms. Mr. Benner earned a B.S. from Purdue University, West Lafayette, Indiana, and a Ph.D. from the University of Minnesota, Minneapolis, both in chemical engineering. He joined Sandia in 1983. Mr. Harris is supervisor of program development in the Center for Computational Engineering. He received a Ph.D. in applied physics from the California Institute of Technology, Pasadena, and joined Sandia in 1976.*

Sandia National Laboratories has developed massively parallel (MP) software that provides high absolute performance and nearly perfect parallel efficiencies. We have used this software on first- and second-generation 1K-processor hypercubes with more than thousand-fold parallel speedups and supercomputer performance. As we developed MP software, we made significant research advances in parallel numerical methods, such as parallel time stepping. The hardware diagnostic tools and parallel graphics algorithms that we have developed and provided to vendors can pinpoint hardware problems and visualize performance in thousand-processor ensembles. Our distributed-computing software will provide supercomputer performance from workstation networks. In collaboration with university and industry researchers in parallel computing, we have proven the concept of distributed parallel supercomputing with parallel computations on a network of VAX® computers. (VAX is a registered trademark of Digital Equipment Corporation.)

Introduction

Parallel computing is an interdisciplinary field that deals with the hardware and software challenges on a system of multiple processors linked by a communication network. The development of parallel algorithms, a collection of step-by-step procedures that comprise an application, is central to parallel computing. We will review recent progress in parallel algorithms and implementation strategies for applications on MP computers. This includes the development of production-quality software (e.g., radar simulation and electron-microscope simulation programs) that run on MP systems such as hypercubes.

Sandia has developed MP software that provides high absolute performance and nearly perfect parallel efficiencies. Parallel efficiency

Panel 1. Abbreviations, Acronyms, and Terms

DEGOM — a public domain molecular structure code
Flow — fluid dynamics code of United Technologies
Fortran — formula translator language
I/O — input/output
LAN — local-area network
LISP — list programming; a flexible symbolic programming language
MIMD — multiple instruction, multiple data
MP — massively parallel
PDE — partial differential equation
RPC — remote procedures call
SDI — Strategic Defense Initiative
SIMD — single instruction, multiple data
TCGMSG — a distributed computing software package of Oak Ridge National Laboratories
teraFLOPS — trillion floating-point operations per second
UDP — unreliable datagram protocol
VLSI — very-large-scale integration
WAN — wide-area network

refers to the computation rate per processor in parallel computation compared to the rate for the same computation when it is executed on only one processor.

A *hypercube* is a three-dimensional cube that is replicated and extended to additional dimensions, as we will discuss later. It is a parallel processor—a computer with many processors that can work on a problem simultaneously. Each processor is connected to its own set of memory chips, forming a *processing element*, or *node*. These are lined up in cube-like patterns of varying dimensions.

A massively parallel computer has many parallel processors. In this paper, *massive parallelism* refers to general-purpose, multiple-instruction, multiple-data (MIMD) systems with 1000 or more autonomous floating-point processors, and to single-instruction, multiple-data (SIMD) systems with thousands of one-bit processors and floating-point coprocessors. Massively parallel computers are often referred to as scalable computers. Scalability involves both parallel architectures and parallel algorithms, which continue to achieve high parallel efficiencies as the number of processors is increased. (See Panel 1 for definitions of abbreviations, acronyms, and terms.)

The Systems. A hypercube, shown in Figure 1, is a recursively defined, logarithmic communications network. A hypercube of dimension 1 has two nodes with one communications link between the nodes; the familiar cube of dimension 3 has eight nodes and three communications links. A hypercube of dimension 10 has 1024 processor nodes, with 10 communications links at each node and a maximum distance of 10 communications links between any two nodes. Sandia has two types of MP architectures that are based on the hypercube network. The MIMD family is represented by the nCUBE 2 and nCUBE/ten hypercubes and the SIMD family by the CM-2X version of the Connection Machine. (nCUBE 2 and nCUBE 10 are trademarks of nCUBE and the Connection Machine is a trademark of Thinking Machines Corporation.) Sandia's nCUBE 2 has 1K processors with 4 Mbytes (megabytes) of local memory per processor, while the nCUBE/ten has the same number of processors but only 0.5 Mbyte of memory per processor. Currently, the CM-2X is configured with 16K single-bit processors, 128 kbytes (kilobytes) of memory per processor, and 512 coprocessors that handle 64-bit floating-point instructions.

The Team. About 50 members of Sandia's staff are pursuing parallel computing topics on the systems described earlier. This interdisciplinary group combines computational scientists and engineers, working with applied mathematicians and computer scientists. We believe that interdisciplinary teams are a strong asset, perhaps even an essential ingredient, to advancing state-of-the-art parallel supercomputing. This team's work broke a major psychological barrier to the use of MP computers in the computational sciences and other fields of endeavor.

Lessons from Highly Parallel Applications

Parallel application development draws, in turn, on continuing research advances in such areas as strategies for mapping problems to processors, methods for dynamic load balance, models and tools for performance

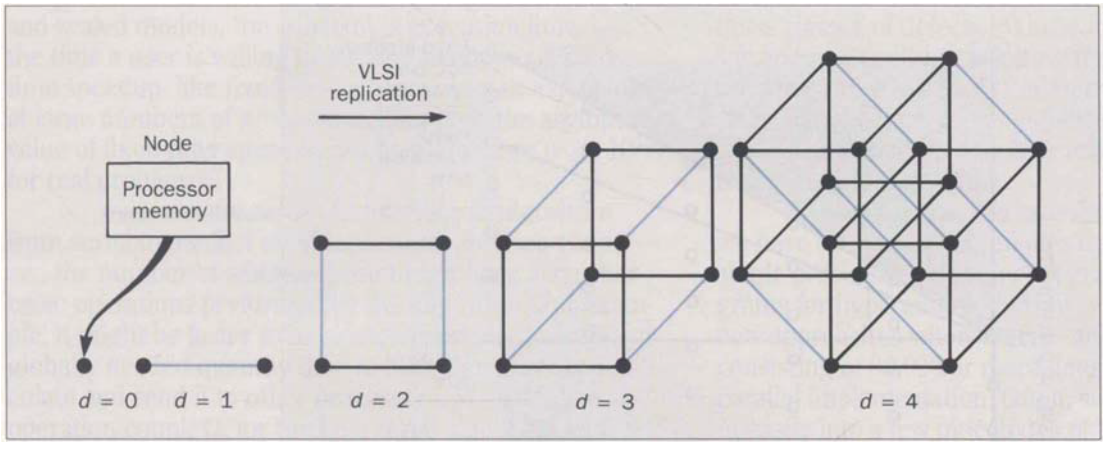

Figure 1. Construction of a hypercube network of dimension *d*. Hypercube multiprocessors with *d* = 11 (2048 processors) have been built; current commercial products are scalable to *d* = 13 (8192 processors).

analysis, and techniques for MIMD programming. Early work at Sandia showed that regular partial differential equation (PDE)-type problems can achieve nearly perfect speedup on 1K processors,[1] even when all input/output (I/O) produced by the parallel application code is taken into account. This is done by taking advantage of static load balancing, regular problem domains, replication of the executable program on every node, and scalability. Sustained computation rates 1.6 to 4 times that of a conventional vector supercomputer were measured for this kind of problem on our first-generation parallel *ensemble*, the 1K-processor nCUBE/ten hypercube.

In early applications of parallel processing, we treated problems by using dynamic load-balancing requirements, global data sets, and third-party application programs too large and complex to replicate on every processor. The performance advantage of this ensemble over conventional supercomputers increased in many cases[2-4] with the size and irregularity of the program. For example, a radar simulation program sustained speeds 7.9 times that of the Cray X-MP supercomputer and 6.8 times that of the Cray Y-MP supercomputer on our first-generation hypercube, including initialization, input, and output phases of the computation. For purposes of comparison, the Cray machines are vector supercomputers with four or eight processors, respectively; each Cray processor has a speed that is 100 times that of the microprocessor on one of the nCUBE 2 nodes. The hypercube is 15 to 25 times more cost-effective than a Cray version using all four or eight of its processors.

In real applications, some of the major performance and cost advantages of the MP approach over conventional vector supercomputers include parallel radar simulation; a Strategic Defense Initiative (SDI) missile and reentry vehicle tracking simulation; some Monte Carlo (random event) simulations; and shock hydrodynamics codes, which model explosions and high-velocity collisions. Moreover, in nearly all cases, the algorithms can probably be used on higher levels of parallelism than the 1K-processor MIMD or 64K processor SIMD levels tested to date.

Given that the parallel processing community is not uniformly experiencing such success, it is important to review our understanding of massive parallelism. Four areas are pertinent:

- The development of performance models, which show that 90- to 100-percent parallel efficiencies are achievable.
- The analysis of redundant operations in a parallel

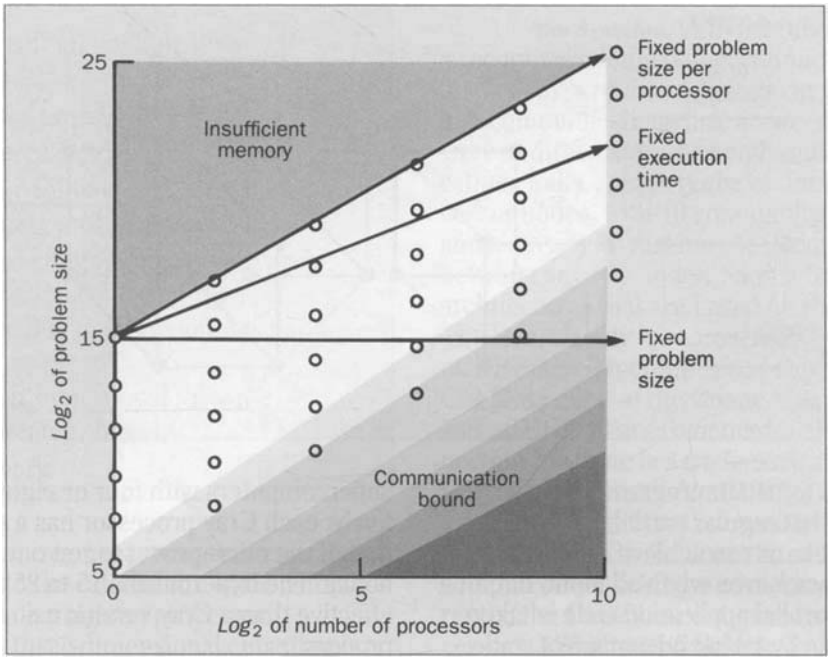

Figure 2. Ensemble computing performance pattern. The choice of a benchmark (fixed overall problem size, fixed execution time, fixed problem size per processor) depends on the region of the performance diagram that is relevant for a given application. We typically sample the performance diagram on a two-dimensional grid, such as that given by the data in this figure.

computation, which lead to inefficiency.

- The use of parallel applications to diagnose hardware and analyze performance.
- The development and implementation of innovative programming strategies used for large parallel applications.

Our understanding of these areas creates opportunities for considering massive parallelism and for developing and using heterogeneous MP systems.

Performance Models. Parallel performance depends on a domain defined by problem size and parallel system size, among other variables. Early in our work, we examined three subsets of this domain (shown in Figure 2): fixed overall problem size,[5] fixed problem size per processor,[6] and fixed execution time.[7] We also routinely explored the more general, two-dimensional domain by gathering performance data at the points shown in Figure 2.

Amdahl's law[5] was stated for a subset of the performance domain called the fixed-size speedup line. On this line, problem size is held fixed and ensemble size is varied. Speedup is then calculated as the ratio of the run time on one processor to that on P processors. For large parallel ensembles, fixing the problem size can be a severe constraint, because the problem must run efficiently when variables occupy a small fraction of each memory.

In practice, a scientific computing problem often scales with (i.e., is within the range of) the available processing power, either memory or speed. Most users will solve larger problems given greater hardware resources. The scaled model assumes the problem size is within the limits of available memory. It is much easier to reach efficient parallel performance than is implied by Amdahl's model, and speedup as a function of the number of processors has no upper limit.

In a third model, intermediate to the fixed-size

and scaled models, the constant is execution time, i.e., the time a user is willing to wait for an answer. Fixed-time speedup, like fixed-size speedup, has an asymptote at large numbers of processors. However, the asymptotic value of fixed-time speedup can be quite large (e.g., 10^{12}) for real problems.

Operation Efficiency. Converting an algorithm from serial to parallel often increases operation count, i.e., the number of additions, multiplications, and other basic operations performed by the algorithm. For example, it might be faster to have each processor calculate a globally needed quantity than to have one processor calculate and send it to other processors. We calculate an operation count, Ω, for the best serial algorithm and another, Ω_p, for the best parallel algorithm. Generally, $\Omega_p \geq \Omega$. Hence, p processors can be 100 percent busy during computation and still be less than p times faster than the best serial algorithm.

Operation efficiency, Ω/Ω_p, an algorithmic consideration, can account for the loss of efficiency in highly parallel applications. Therefore, operation efficiency is an important issue for the analysis and development of highly parallel algorithms. Modeling Ω/Ω_p provides an analytical model of the tradeoff between communications overhead and redundant operations, which can be used to tune parallel performance. Operation efficiency can help us avoid overly optimistic expectations of performance gains to be achieved by introducing parallelism.[8]

Hardware Diagnosis. The hardware of MP ensembles also can cause subtle efficiency losses. At times, we have observed anomalies in the performance of a wave mechanics (inviscid fluid flow) application with greater than 99-percent parallel efficiency on various sized subcubes of our hypercubes. For example, these anomalies could occur on one half of the hypercube but not on the other half.

An almost perfect efficiency was found to be reduced to 90 to 97 percent in a subcube containing defective processor nodes. To date, we have identified three classes of defects in these nodes that slow processor operation without affecting the results.[1] A version of the wave mechanics application has been fine-tuned as a "slow node" hardware diagnostic for finding and replacing defective nodes. This technology was successfully transferred to the vendor.

MIMD Programming Strategies. In the past decade, we have seen many techniques developed for turning single-processor computer programs into parallel programs for hypercubes.[9,10] However, we must consider new approaches when we take an application package consisting of 30,000 or more lines of code and develop a parallel implementation. Often, such simulations do not fit easily into a few megabytes of local memory. One solution to this problem is to parcel portions of the application package onto separate subsets of the processors, rather than put all the executable programs in the package on each processor. Different programs in the application are executed simultaneously and asynchronously on different portions of the hypercube. They then exchange data as needed. Figure 3 shows two examples of a heterogeneous programming strategy.

Heterogeneous programming uses a MIMD computer in a general MIMD fashion. In contrast, the authors of Reference 1 observed that the parallel applications they described were SIMD-like and would run very well on the Connection Machine.

Figure 3 shows a general MIMD approach in which three or more processes are executing asynchronously. For the radar simulation[3], four processes are involved: a dynamic load-balance process, a multibounce radar ray-tracing process, a radar imaging process, and a process that performs a global collection of the image (if needed). Not shown are two additional, vendor-supplied library processes: a graphics process, which executes on a set of I/O processors; and a host utility to handle hypercube I/O, which executes on the front-end processor of a Sun Workstation® system. (Sun Workstation is a registered trademark of Sun Microsystems, Inc.) Similarly, an SDI application[9] that involves tracking tens or

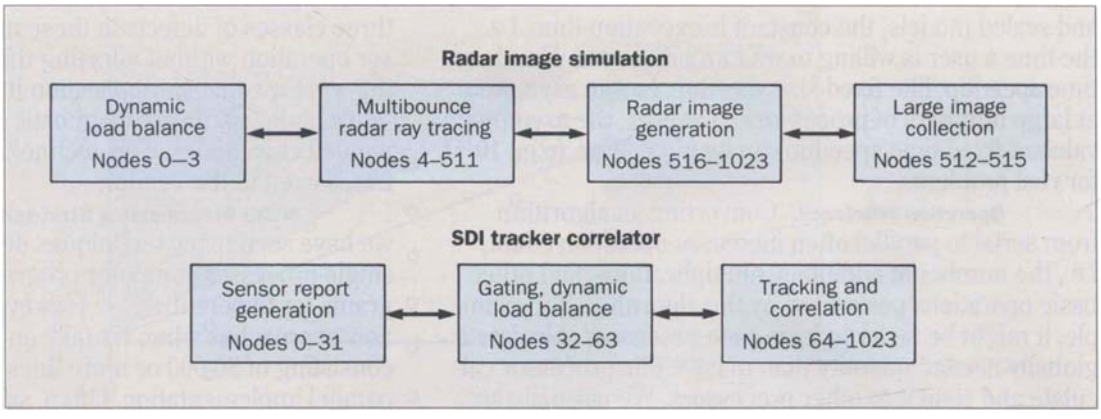

Figure 3. Heterogeneous implementations of two parallel applications on the hypercube. Three to four parallel codes run simultaneously and cooperatively. Maintaining the independence of the original codes in the application packages simplifies code development.

hundreds of thousands of objects through space has been implemented as a set of three cooperating node programs.

MP Systems. There is an ongoing debate about what massive parallelism means. Purists regard massive parallelism as a large collection of processors that form a statistical ensemble. This belief enables purists to disregard the efficiencies or the activities of individual processors. By this definition, most or all MIMD parallel applications to date have not been massively parallel because the computations are highly synchronous and proceed in lockstep (e.g., in typical grid-based—i.e., finite difference or finite element—simulations).

Now, contrast the lockstep MIMD paradigm with the heterogeneous implementations outlined earlier. The heterogeneous applications have various collections of processors primarily concerned with their own parallel subtasks. The computations are loosely synchronous and, to a large extent, do not take into account the efficiencies of individual processors. An efficiency of 100 percent is highly undesirable for the process that handles dynamic load balancing, because load-balance nodes would be saturated and unable to keep up with work requests from other processors. Efficiencies of 20, 30, or 50 percent are acceptable for nodes that execute the load balancer, as long as most processors are working at close to 100-percent efficiency. In this sense, the heterogeneous implementation truly uses the MIMD computer as an MP ensemble.

Heterogeneous MP Systems. Any use of heterogeneous software offers an opportunity to use heterogeneous hardware. For example, given the heterogeneous implementation of the SDI algorithm in Figure 3, we can quantify the computational workload within different tasks and, therefore, the communications bandwidth needed. We can also assess the computing and communications capabilities needed to take a distributed-computing approach to this problem, so that one task is done on a space platform, while others are done on the ground.

Are MIMD processors needed for all phases of a heterogeneous computation? On a given application, the SIMD paradigm can probably be used below the high level at which the MIMD paradigm is introduced. In addition, much can be done at the processor level within a heterogeneous implementation by using heterogeneous nodes. For example, for a critical task, we can use some nodes with larger memories to reduce the total number of processors devoted to that task.

Given a good heterogeneous software implementation, we can propose heterogeneous hardware systems to run the computations efficiently. Conversely, it would be risky to build such a system without having experi-

ence with heterogeneous implementations, which can be developed on a more general-purpose homogeneous hardware system. In summary, a heterogeneous implementation on a homogeneous machine, coupled with a careful performance analysis, may be an attractive and prudent path to future general- and special-purpose heterogeneous systems.

Novel Parallel Algorithms

The area of parallel algorithms can be roughly divided into two topics: numerical and nonnumerical algorithms. Numerical algorithms typically arise in applied mathematics, computational science, and engineering problem solving. Nonnumerical algorithms most often occur in discrete mathematics and computer science. We will briefly discuss come advances in parallel numerical methods, as well as dynamic load-balance and parallel graphics methods, which will serve as examples of parallel nonnumeric methods.

Parallel Numerical Methods. During the 1980s, much of the activity in parallel computing focused on individual algorithms rather than complete applications. Cellular automata, adaptive precision numerical methods, and highly parallel variants of the multigrid method (a method that uses fast, inaccurate calculations on a few grid points to accelerate the accurate solution of a problem on many grid points) emerged as areas of interest in SIMD computing. Similarly, there was considerable experimentation in MIMD computing with asynchronous numerical methods, as well as standard matrix methods.

On MIMD machines, the classic multigrid method has measured parallel efficiencies of 85 percent for two-dimensional problems on 1K processors, and about 70 percent for three-dimensional problems on 1K processors,[11,12] exceeding expectations for methods that have a significant serial nature.

Another method that has emerged is parallel time stepping.[13] In parallel time stepping, different processors or sets of processors iterate on different time steps simultaneously, which allows us to extract a 4- to 16-fold increase in parallelism in a problem over and above the usual spatial parallelism. Parallel time stepping does not, at present, dramatically increase parallelism, but it does considerably expand the range of applications that can use MP systems efficiently.

Additional work, both experimental and theoretical, has been done on various asynchronous methods,[14] including proofs of convergence for some of the most interesting ones. MP algorithms have been developed for linear algebra for dense matrices,[15] integer sieving[16] and factoring,[17] molecular dynamics,[18] fast Fourier transforms and other signal processing algorithms, and applications of standard[19] and novel[20] Monte Carlo methods.

Dynamic Load Balance. Some MP applications reach efficiencies close to 100 percent on 1K processors. In these applications, the same amount of work is assigned to each processor. Time is saved because the allocation of work and storage to each processor can be computed once at the beginning of the run and applied to the entire problem. Also, the problem domains are regular and can be readily divided among the hypercube processors using conventional techniques.

For static load balance of problems defined on irregular domains, we use innovative mapping methods.[21] These include graph-based and binary decomposition methods. Graph-based methods operate on a graph that describes the interrelationship of work units. Binary decomposition methods repeatedly cut the workload in half to produce many small workloads, one for each processor. Although significant progress has been made in parallel mapping methods, it may still take almost as long to solve the mapping problem as it does to solve the physical problem of interest.

In applications where the amount of work in regions of the domain varies as the applications are running, dynamic load balancing schemes are needed (see Figure 4). A host-dependent, dynamic algorithm, such as the traditional master-slave method, succumbs to a communications bottleneck and does not scale to more than a few processors. For example, if each processor handles

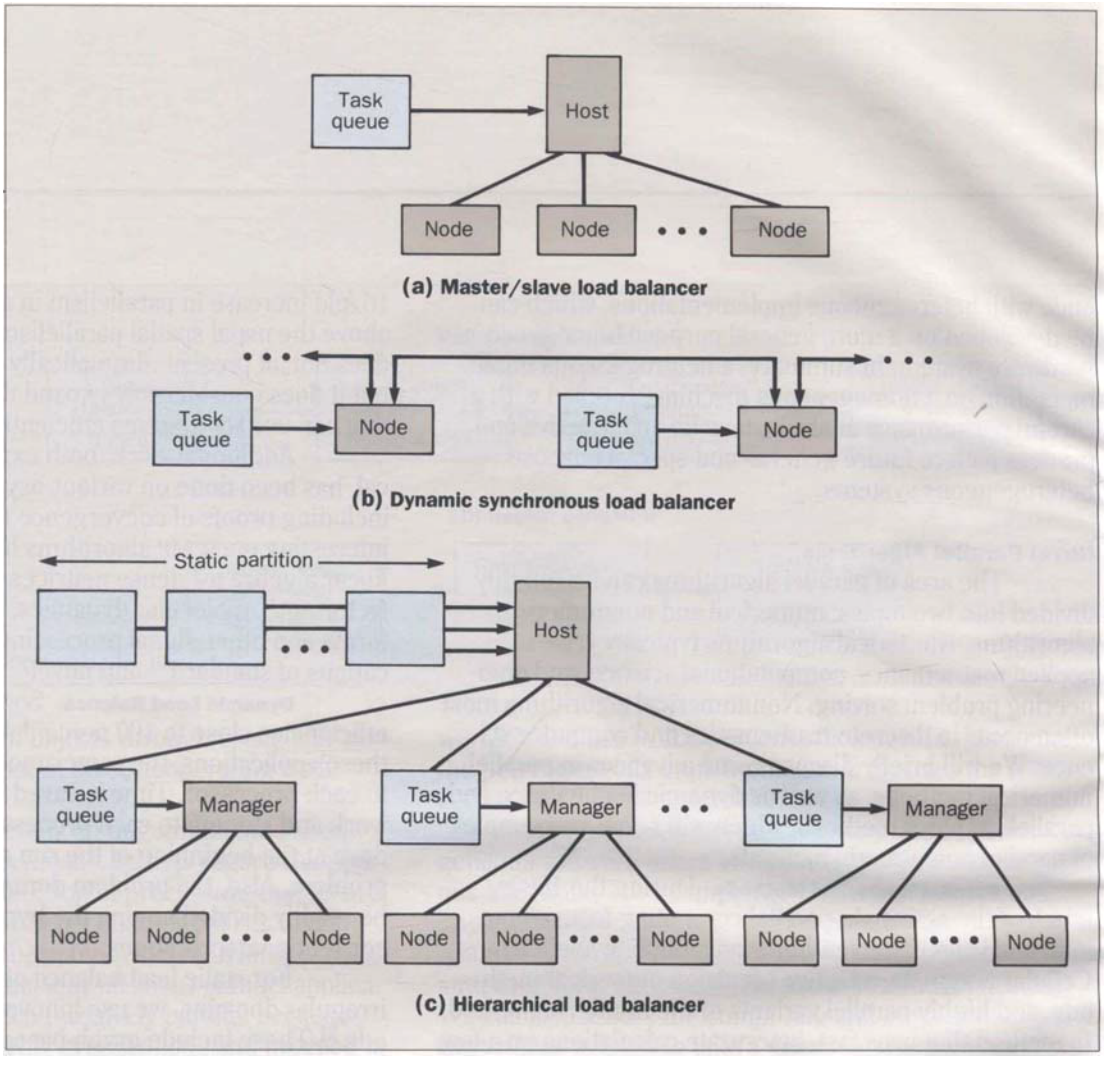

Figure 4. Three dynamic load-balance methods. The limitations of the (a) classic master-slave scheme are overcome by recent innovations, such as (b) dynamic synchronous and (c) hierarchical load-balance methods.

N tasks per second, then the host has to perform $O(NP)$ operations and handle $O(NP)$ messages per second on a P-processor system. Therefore, the load-balance algorithm should execute only on the nodes.

Figure 4 shows three node-based, dynamic load-balance methods. A hierarchical load balancer[3] has been developed for dynamic simulations without natural time-step synchronization (e.g, the radar image and SDI simulations outlined earlier).

The hierarchical method is an extension of the classic master-slave method, in which the master processes are located on nodes rather than on a host processor. A third method was developed to allocate work evenly in synchronous transient simulations[22] involving independent spatial variables (e.g., particle simulations). Here, load-balance operations can be distributed locally among all the nodes of the hypercube, rather than being performed by "master" and "slave" processors.

Three key issues in the design of any load-balancing algorithm are time, storage, and effectiveness.[22] *Time* for the load-balancing operation should not exceed the time gained by reducing imbalance in the application. *Storage* needed for the algorithm should be kept low to make the rest of the simulation scalable. The distributed approach can balance efficiently with only $O(1)$ additional memory. *Effectiveness* is maximized by the distributed load balancer, which is based on binary domain decomposition. It produces ideal balance at

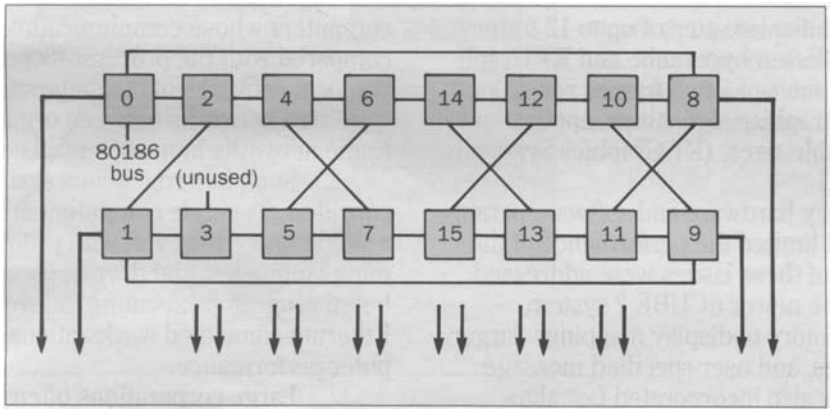

Figure 5. A multiprocessor I/O subsystem. Nonuniformity of interprocessor communications within the subsystem, and between the subsystem and the MP computer, makes programming such subsystems difficult.

synchronization points in an application. The algorithm can be used most effectively in any problem where the cost to quantify the amount of work in each time step is logarithmic to the number of processors.

Parallel Graphics Algorithms. The ability of MP systems to generate many megabytes or even gigabytes of useful data per second is a driving force behind research on and development of high-speed interfaces and networking for supercomputers. Indeed, the interfaces themselves must be highly parallel (i.e., tens to hundreds of processors) to match the potential I/O capability of current and near-term MP systems. We must address many hardware issues associated with parallel interfaces to MP systems to achieve high-performance I/O, such as:

- The interconnection network between interface nodes, which dictates how efficiently nodes can communicate and synchronize with one another
- Mapping incoming or outgoing data to interface node memories, which defines the amount of data movement necessary to conform to standard communications protocols
- The interconnection network between interface nodes and the nodes of the MP system, which controls how efficiently and reliably data can be transferred to the interface.

Figure 5 presents a case study of a multiprocessor I/O interface: the graphics subsystem on the nCUBE/ten hypercube. This system has 16 I/O nodes, which are distinct from the 1024 nodes in the hypercube ensemble. A thin line denotes a communications link between 2 of the 16 I/O nodes. A thick line represents the 8 communications links between an I/O node and 8 of the 1024 nodes in the hypercube. The first generation of vendor software for the subsystem did not support their quoted 30-frame-per-second display rate; an image required about 3 seconds per frame image from 1024 nodes.

We developed four parallel graphics algorithms on this subsystem for visualization of complex problems in PDE simulations, radar simulation, and other large applications.[23] We explored parallel graphics hardware and defined software limitations. Some of the algorithmic techniques we developed for dealing with system constraints included:

- Multistage routing of graphics data through the large hypercube ensemble to keep message buffers from overflowing
- Explicit use of nonhypercube mappings for routing between the ensemble and graphics device
- Tree algorithms for fast synchronization and data fan-in and fan-out between graphics nodes
- Synchronization between the ensemble and the graphics nodes.

As a result, we achieved display rates of up to 12 frames per second on an nCUBE/ten hypercube and RT Graphics System, with typical rates of 2 to 5 frames per second across a wide range of graphics algorithms, applications, image sizes, and ensemble sizes. (RT Graphics System is a trademark of nCUBE.)

We identified key hardware and software parameters that governed and limited the performance of the graphics system. Most of these issues were addressed later by the vendor in the newer nCUBE 2 system, including simplified memory-to-display mappings, larger graphics node memories, and user-specified message buffer sizes. The vendor also incorporated fast algorithms into the library software for the new system, with sustained transfer rates of 50 Mbytes per second measured between the hypercube and the graphics system.

Many of the algorithmic techniques we applied to parallel graphics are useful in other I/O and communications tasks. For example, a parallel disk system might consist of 16 or more disks served by 16 processor nodes on an interface board. Application program issues, such as message buffer limitations and message routing considerations, are the same for both systems. Our hope is that the lessons learned in parallel graphics will further the development of high-speed, highly parallel network and disk I/O interfaces.

Distributed Parallel Supercomputing

Many organizations own a supercomputer without even knowing it. They have a distributed supercomputer—an interconnected network of personal computers, workstations, and mainframes with total processing power greater than many conventional supercomputers. Some of these machines may also connect local high-speed lines to the local- or wide-area network.

A distributed computer system consists of geographically dispersed processors that communicate primarily through messages (see Figure 6). This definition is similar to the one given by Cristian and Skeen,[24] but it focuses more on computations performed on networks of computers whose communications throughput is slow compared with the processor's performance. Included are local networks of computers used by individuals, wide-area networks between organizations, and mainframe networks in most central computer centers.

Such heterogeneous systems are not usually considered a single computer capable of cooperating on a single task. However, with proper algorithms, programming languages, and development tools, classes of distributed programs executing on even a modest collection of Ethernet-connected workstations can achieve supercomputer performance.

Large corporations often have networks that connect many hundreds or even thousands of processors. If an MP computer is defined as one with at least a thousand nodes,[1] most of the MP computers in the world are network-based distributed computers as opposed to mainframe multicomputers. Few, if any, of these MP distributed computers are being used as supercomputers because it is difficult to program applications on them or port uniprocessor applications to them. The remainder of this section describes the status of software that helps develop parallel applications for experimental distributed systems. Such systems are potentially massively parallel. Currently, they consist of 10 to 100 processors.

Distributed Computing Software. A distributed computer system can be programmed as serial or parallel. Most of this paper deals with parallel execution. However, serial execution on a network of specialized processors will often outperform a general-purpose processor.

We can partition a complex simulation so that tasks that vectorize efficiently are executed on a vector computer, data is manipulated with database servers, and graphics rendering is performed on specialized graphics hardware. The simulation may run more efficiently on a network of processors, even if the network resources are used in series, than it would on a general-purpose processor. Of course, if we can overlap the simulation with network resources, the performance is increased. The potential of network-based distributed computing lies in

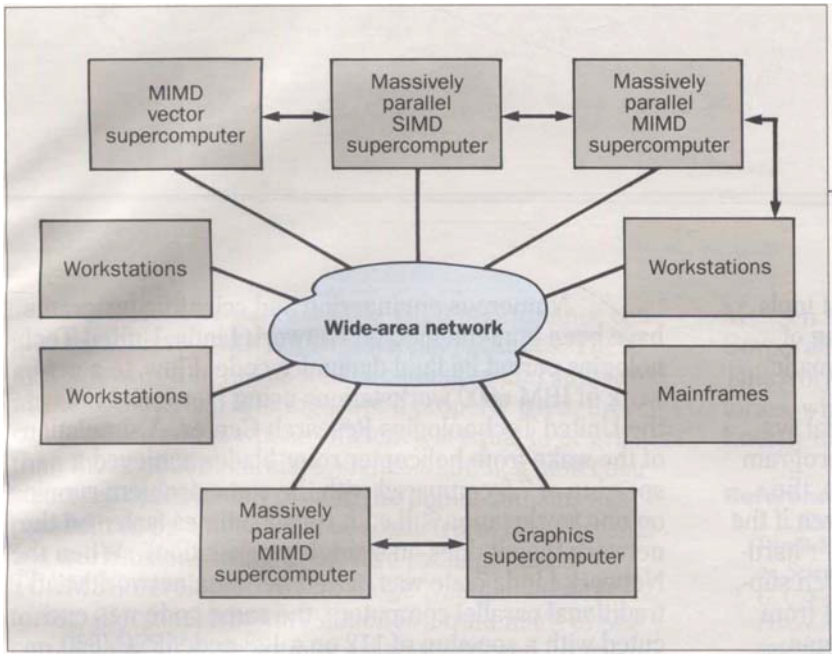

Figure 6. Distributed massively parallel computing. In the future, massively parallel computers may be viewed as one of the many resources in a distributed system that is itself massively parallel.

parallel use of network processors.

Distribution of tasks across a network of loosely coupled processors requires a computing environment that can synchronize processors. Several commercial packages exist, including:

- Network Linda (trademark of Scientific Computing Associates)
- Strand88 (trademark of Artificial Intelligence Limited of Great Britain)
- Express (trademark of Parasoft Corporation)
- Cronus (trademark of BBN Systems and Technologies)
- Network Computing System (NCS) (trademark of Hewlett-Packard)
- Open Network Computing (ONC) (trademark of Sun Microsystems)
- PIPES (trademark of PeerLogic).

There are also many noncommercial packages for network computing, including:

- Distributed Computing Environment (DCE) (trademark of Open Software Foundation)
- PICL/PVM (product of Oak Ridge National Laboratories)
- TCGMSG (product of Argonne National Laboratories).

The SPAWN project (project of Xerox PARC) is an experimental, high-performance network operating system. These lists are by no means exhaustive.

Each of the systems listed has distinctive strengths and weaknesses. The systems range in complexity from simple subroutine packages that improve message passing between nodes, to full development environments including communications-sensitive simulators, syntax-sensitive editors, and symbolic debuggers. The variety allows software developers to choose the package that best fits an application. Unfortunately, the developer must adopt a system-specific view of the role of the processes and data distribution in the program.

At the lowest level, all systems except PIPES are based on remote procedures calls (RPCs). Because each system has its own program paradigm, the implementation and use of RPCs are distinct in each system. For instance, Network Linda uses synchronous RPCs, which bind a client process to a single server process for the duration of the procedure. The Cronus system uses asynchronous RPCs, which allow multiple, concurrent interactions among clients and servers.[25] Software engineers must take these differences into account if application performance is to be optimized. The system-specific nature of the resulting code restricts portability between software systems.

The intellectual effort and time necessary to generate a distributed application depends both on the system chosen and the application. Some of the software

systems mentioned earlier include development tools that help decompose serial programs and testing of parallel applications. To date, there are no automatic parallelizing compilers for distributed systems.

All noncommercial and most commercial systems support multiple platforms. As soon as a program has been parallelized using a particular package, the resultant code is portable between networks, even if the networks have different topologies and computer hardware from different vendors. Many packages even support networks containing a mixture of hardware from more than one vendor. Of the commercial systems, researchers have used the first three on local-area networks (LANs), along with computer hardware from a variety of vendors.

Parallel Performance on LANs. The Network Linda distributed environment[26] was an early focus of our work. Network Linda is not a language; rather, it is a set of objects and operations on those objects that are injected into existing languages, including Fortran and LISP. The first implementation was based on C language and ran on the experimental SNet machine developed at AT&T Bell Laboratories.[27]

A current research project at Bell Laboratories involves designing a hardware implementation of the basic Network Linda operations.[28] Software implementations on shared memory multiprocessors include those available on Encore and Sequent systems, as well as distributed memory multiprocessors like the Intel iPSC. (Sequent is a trademark of Sequent Computer Systems, Inc., and iPSC is a trademark of Intel Corporation.)

Network Linda is an extension of the distributed memory version using the unreliable datagram protocol (UDP) for interprocessor communications. The commercial version of Network Linda is limited to homogeneous networks of Sun Workstations systems or Silicon Graphics Workstations. A version that can support a heterogeneous mix of UNIX® system-based platforms is being tested. (UNIX is a registered trademark of UNIX System Laboratories, Inc.)

Numerous engineering and scientific programs have been implemented on Network Linda. United Technologies ported its fluid dynamics code, Flow, to a network of IBM 6000 workstations using Network Linda at the United Technologies Research Center. A simulation of the wake from helicopter rotor blades achieved a speedup of 7.5 compared with the same problem running on one workstation[29] (i.e., it runs 7.5 times faster on the network than it does on a single workstation). When the Network Linda code was ported between networks and traditional parallel computers, the same code was executed with a speedup of 112 on a 128-node iPSC/860.

Researchers at Yale University and the University of Illinois used a network of 15 SPARCstation workstations to compute the nonbonded energy terms in a molecular dynamics simulation for a 3634 atom segment of a large photosensitive protein. They achieved a speedup of 12 over a single SPARCstation workstation. (SPARCstation is a trademark of SPARC International.)

With a larger network of 40 SPARCstation workstations, Yale University researchers used the Rayshade code for rendering photorealistic scenes to achieve a speedup of 30 on Network Linda.[30] The same code, which does not perform well on a Cray supercomputer, is about a factor of 10 times faster on the network than on a single Cray 2 processor.

Parallel Performance on WANs. Whiteside[31] describes a wide-area network (WAN) used for distributed processing. An early version of the Network Linda software in a rocket-plume sensitivity calculation achieved 2.40 Cray 1 equivalents and a speedup of 10.6 using a network of 14 VAX 8000-class computer processors. The network includes machines at Sandia sites at Albuquerque, New Mexico, and Livermore, California. In these calculations, the computation time is much greater than the communications time, resulting in high parallel efficiency. In contrast, a speedup of only 2.5 was achieved using four VAX computer processors for matrix inversion, a task that depends on frequent interprocessor communications.

Noncommercial Packages. High performance can also be achieved with noncommercial packages. These systems are often not as well documented and supported as the commercial offerings, but if properly used, they can produce codes whose efficiency is equal to or greater than those of commercial systems. DEGOM, a molecular structure optimization code in the public domain, was parallelized on a network that consisted of five Personal Iris workstations using a software package called TCGMSG developed by Oak Ridge National Laboratories. (Personal Iris workstation is a trademark of Silicon Graphics.) A speedup of 4.4 was achieved for calculations consisting of 100 structures with 52 atoms per structure, at about 30 percent of the speed at which a vectorized version of DEGOM runs on a single processor of a Cray X-MP supercomputer. This is comparable to the best performance reported on similar molecular structure computations using commercial network packages. The DEGOM tool kit supports heterogeneous networks, so we have added a Stardent Titan to the experimental network. (Stardent is a trademark of Stardent Computer Inc.) Later, we may add a SPARCstation workstation.

Conclusions

Massively parallel computers have come of age as supercomputers. The number of MP applications will increase steadily, and the fraction of those that will move into production status will also rise, particularly as the I/O and network interface bandwidth of MP machines becomes more balanced with the internal bandwidth of these architectures. In turn, the degree of parallelism employed in I/O and network subsystems will increase dramatically, and experimentation with heterogeneous MP systems, particularly highly parallel distributed systems, will begin in earnest.

Acknowledgments

Special thanks go to our many colleagues whose work we have briefly summarized. We also thank Tim Mattsen (Strand Corporation) and Leigh Kagan (SCA Corporation) for sharing their performance data with us. This work was performed at Sandia National Laboratories, which is operated for the U.S. Department of Energy under Contract Number DE-AC04-76DP00789.

References

1. J. L. Gustafson, G. R. Montry, and R. E. Benner, "Development of Parallel Methods for a 1024 Processor Hypercube," *SIAM Journal on Scientific and Statistical Computing*., No. 9, 1988, pp. 609–638.
2. G. C. Fox, "1989—The First Year of the Parallel Supercomputer," *Proceedings of the Fourth Conference on Hypercube Concurrent Computers and Applications*, Monterey, California, March 1989, Golden Gate Enterprises, Los Altos, California, pp. 1–37.
3. J. L. Gustafson et al., "A Radar Simulation Program for a 1024 Processor Hypercube," *Proceeding of Supercomputing '89*, Reno, Nevada, November 1989, ACM Press, New York, 1989, pp. 96-105.
4. S. S. Dosanjh, ed., *MPCRL Research Bulletin*, Sandia National Laboratories, Albuquerque, New Mexico, February 1991.
5. G. Amdahl, "Validity of the Single-Processor Approach to Achieving Large-Scale Computer Capabilities," *American Federation of Information Processing Society Conference Proceedings*, No. 30, 1967, pp. 483-485.
6. J. L. Gustafson, "Reevaluating Amdahl's Law," *Communications of the ACM*, No. 31, 1988, pp. 532-533.
7. R. E. Benner, G. R. Montry, and J. L. Gustafson, "A Structural Analysis Algorithm for Massively Parallel Computers," *Parallel Supercomputing: Methods, Algorithms and Applications*, Chapter 10, G. F. Carey, ed., Wiley & Sons, New York, 1989, pp. 115-134.
8. D. H. Bailey, "Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers," *RNR Technical Report RNR-91-020*, NASA Ames Research Center, Moffett, California, 1991.
9. J. L. Tomkins and John P. VanDyke, "Implementation of Midcourse Tracking and Correlation on Massively Parallel Computers," *Proceeding of Supercomputing '91*, Albuquerque, New Mexico, November 1991, ACM Press, New York, 1991, to be published.
10. G. C. Fox, et al., *Solving Problems on Concurrent Processors*, Vol. I, Prentice Hall, Englewood Cliffs, New Jersey, 1988.
11. R. S. Tuminaro, "A Highly Parallel Multigrid-Like Method for the Solution of the Euler Equations," *SIAM Journal on Science and Statistical Computing*, Vol. 13, Issue 1, January 1992, to be published.
12. D. E. Womble and B. C. Young, "Multigrid on Massively Parallel Computers," *Proceedings of the Fifth Distributed Memory Computer Conference*, Charleston, South Carolina, April 1990, IEEE Computer Society Press, Los Alamitos, California, 1990, pp. 559-563.
13. D. E. Womble, "A Time Stepping Algorithm for Parallel

Computers," *SIAM Journal on Scientific and Statistical Computing*, No. 11, 1990, pp. 824-837.
14. D. E. Womble, "The Performance of Asynchronous Algorithms on Hypercubes," *Report SAND88-2714*, Sandia National Laboratories, Albuquerque, New Mexico, 1988.
15. M. P. Sears, "Linear Algebra for Dense Matrices on a Hypercube," *Proceedings of the Fifth Distributed Memory Computer Conference*, Charleston, South Carolina, April 1990, IEEE Computer Society Press, Los Alamitos, California, 1990, pp. 317-320.
16. G. R. Montry, ""Massively Parallel Mathematical Sieves," *International Journal of Supercomputer Applications*, Vol. 3, Issue 1, Spring 1989, pp. 59-74.
17. D. B. Holdridge and J. A. Davis, ""Factoring Very Large Numbers Using a Massively Parallel Computer," *Proceedings of the Fourth Conference on Hypercube Concurrent Computers and Applications*, Monterey, California, March 1989, Golden Gate Enterprises, Los Altos, California, 1989, pp. 1089-1091.
18. S. J. Plimpton, ""Molecular Dynamics Simulations of Short-Range Force Systems on 1024-Node Hypercubes," *Proceedings of the Fifth Distributed Memory Computer Conference*, Charleston, South Carolina, April 1990, IEEE Computer Society Press, Los Alamitos, California, 1990, pp. 478-483.
19. A. D. Romig, Jr., et al., ""Application of Parallel Computing to the Monte Carlo Simulation of Electron Scattering in Solids: A Rapid Method for Profile Deconvolution," *Microbeam Analysis—1990*, San Francisco Press, Inc., San Francisco, California, 1990, pp. 275-280.
20. J. M. DeLaurentis and L. A. Romero, "A Monte Carlo Method for Poisson's Equation," *Journal of Computational Physics*, Vol. 90, 1990, pp. 123-140.
21. C. T. Vaughan, ""Structural Analysis on Massively Parallel Computers," *Proceeding of the Conference on Parallel Methods in Large-Scale Structural Analysis and Physics Applications*, Norfolk, Virginia, February 1991, in press.
22. K. M. Dragon and J. L. Gustafson, "A Low-Cost Hypercube Load Balance Algorithm," Sandia Report SAND88-2427, Sandia National Laboratories, Albuquerque, New Mexico, March 1991.
23. R. E. Benner, "Parallel Graphics Algorithms on a 1024-Processor Hypercube," *Proceedings of the Fourth Conference on Hypercube Concurrent Computers and Applications*, Monterey, California, March 1989, Golden Gate Enterprises, Los Altos, California, 1989, pp. 133-140.
24. I. Cristian and D. Skeen, "Foreword: Special Issue on Distributed Systems," *IEEE Transactions on Software Engineering*, Vol. SE-13, 1987, p. 1.
25. J. Koblelus, "LANs Bid for Distributed Power," *Network World*, December 3, 1990, p. 1.
26. D. Gelernter, "Generative Communication in Linda," *Technical Report YALEU/DES/RR-294*, Yale University, Department of Computer Science, November 1983.
27. N. Carriero and D. Gelernter, "SNet's Linda Kernel," *ACM Transactions on Computer Systems*, Vol. 1, 1986, pp. 110-129.
28. S. Ahuja et al., "Matching Language and Hardware for Parallel Computation in the Linda Machine," *IEEE Transactions on Computing*, Vol. 37, 1988, pp. 921-929.
29. *Proceedings of the 2nd Annual UTECA Conference*, Springfield, Massachusetts, April 29–May 2, 1991, United Technologies, 1991.
30. R. Bjornson, C. Kolb, and A. Sherman, "Ray Tracing with Network Linda," *SIAM News*, Vol. 24, No. 1, 1991, pp. 10-11.
31. R. Whiteside and J. Leichter, "Using Linda for Supercomputing on a Local Area Network," Sandia Report SAND 88-8818, June 1988.

(Manuscript received May 13, 1991)