ENABLING DEPARTMENT-SCALE SUPERCOMPUTING[*]

DAVID S. GREENBERG[†], WILLIAM E. HART[‡], AND
CYNTHIA A. PHILLIPS[§]

Abstract. The Department of Energy (DOE) national laboratories have one of the longest and most consistent histories of supercomputer use. We summarize the architecture of DOE's new supercomputers that are being built for the Accelerated Strategic Computing Initiative (ASCI). We then argue that in the near future scaled-down versions of these supercomputers with petaflop-per-weekend capabilities could become widely available to hundreds of research and engineering departments. The availability of such computational resources will allow simulation of physical phenomena to become a full-fledged third branch of scientific exploration, along with theory and experimentation. We describe the ASCI and other supercomputer applications at Sandia National Laboratories, and discuss which lessons learned from Sandia's long history of supercomputing can be applied in this new setting.

Key words. Accelerated Strategic Computing Initiative (ASCI), high-performance computing, supercomputing, interconnection networks, SANs, terascale computing, simulation of physical systems, distributed-memory computing, Pentium-Pro, parallel operating systems.

1. Introduction. The Department of Energy (DOE) National Laboratories have one of the longest and most consistent histories of supercomputer use. From some of the earliest computers used at Los Alamos, to Cray vector supercomputers in the 70's and 80's, to nCUBE and Connection Machine systems in the late 80's, the national labs have required and obtained the biggest, fastest computers available. Through the 90's and 00's, the national labs have and will continue to commission development of ever more powerful machines.

The national defense-program laboratories – Sandia, Los Alamos, and Lawrence Livermore National Laboratories – forsee a long term need for teraflop computing power to successfully achieve their missions in maintaining the safety, reliability, performance, and availability of the nuclear stockpile. On September 24, 1996, President Clinton signed the Comprehensive Test Ban Treaty, agreeing to a zero-yield test ban [33]. Since the United States is also reducing its manufacturing base for nuclear weapons components, the national laboratories are faced with the challenge of maintaining an aging stockpile without the ability to use the ultimate test of functionality. President Clinton's vision was to replace testing with *science-based stockpile stewardship*, meaning virtual prototyping of weapons components, and computer simulation of aging effects, to be coupled with statistical data

[*]This work was supported in part by the United States Department of Energy under Contract DE-AC04-94AL85000. Sanda is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy.

[†] `dsgreen@cs.sandia.gov`; Sandia National Labs, Albuquerque, NM 87185-1110.

[‡] `wehart@cs.sandia.gov`; Sandia National Labs, Albuquerque, NM 87185-1110.

[§] `caphill@cs.sandia.gov`; Sandia National Labs, Albuquerque, NM 87185-1110.

gathered from evaluation (dismantlement) of existing weapons (without detonation).

To achieve this end the *Accelerated Strategic Computing Initiative (ASCI)* [1] began before the test ban was signed, and is expected to continue through 2010. The program funds research and development of the algorithms, applications, hardware, systems software and tools needed to implement science-based stockpile stewardship. In Section 2, we summarize the architectures of the new massively parallel machines under development for the ASCI program.

We believe the lessons learned from the supercomputing efforts within DOE have applicability far beyond the scope of ASCI supercomputing. In Section 3 we argue that systems with supercomputer-level performance (though not yet ASCI-level performance) could soon become available to the *department-scale* research group. That is, by tightly networking commodity components, academic and industrial research departments should be able to afford petaflop/weekend performance by the year 2000. These machines could be built incrementally with minimum funding impact by using "Stone Soup" tactics. In analogy with the children's story, each researcher who wants to use the department's supercomputer will contribute something to the pot: a few more processors, some interconnect, etc.[1]

We review some major supercomputer applications at Sandia National Laboratories (Section 4) to illustrate the capabilities of these machines. We argue that the general methods computational scientists use at Sandia to achieve maximum performance on these machines are generally applicable, particularly for distributed memory machines like the ASCI Red machine.

Section 5 describes some of the lessons learned at Sandia as we have advanced from prototype high-performance computers such as the nCUBE and the Paragon to ASCI-class machines. Some of these lessons can be applied to these new mini-supercomputers, but in some cases there is still much to be learned. In particular, we consider issues of the usage model, programming model, resource management, data movement, system reliability, and code evolution. Section 6 offers some concluding remarks.

2. ASCI supercomputers. In the 90's researchers at various DOE laboratories used many high-performance machines including Paragons, Cray T3Ds, SP2s, nCUBEs, and CM5s. From 1994 to 1997, the primary machine for large simulations at Sandia National Laboratories has been an Intel Paragon. This machine has over 1800 nodes, each consisting of two i860XP processors, which operate at 75 megaflops (MF) each. When one processor on each node is used as a communication coprocessor, as per the original design, this yields a peak performance of 140 gigaflops (GF). Sandia enabled the second processor to be used for computation, though hampered by low memory bandwidth. This allowed some applications to

[1]Recently several researchers at Oak Ridge National Laboratory began an attempt to apply this model literally by collecting equipment scheduled for reapplication [24].

exceed the advertised peak performance of the machine. The nodes are arranged in a 16×120 mesh, with 5 I/O columns in the middle. Communication links can move 200 megabytes per second (MB/sec) in each direction. The machine has 37 gigabytes (GB) of RAM and 330GB of disk space.

At Sandia, we have recently installed the *ASCI Red* machine. Sandia and Intel have expanded the ideas proven successful in the Paragon to create a commodity-based supercomputer. Though the CM-5 used SPARC processors for nonfloating point computation [21], and the Cray T3D uses alpha chips [5], this is the first machine where true supercomputing performance for scientific computing is delivered by processors that will also be used in millions of PCs. Where the Paragon used the end-of-the-line i860 processor, an embedded processor, the ASCI Red machine uses the mainstream Pentium-pro$^{\mathrm{TM}}$ processor. Over 9000 Pentium-pros$^{\mathrm{TM}}$, each of which provides 200MF peak, are tightly integrated to produce a total peak performance of 1.8 teraflops (TF). The machine sustained 1.3TF on the MPLinpack benchmark in June, 1997.

The processors are packaged into dual-processor nodes using standard PC/server chip sets. The dual processors can either be used to perform shared-memory computing or one can serve as a communication co-processor. In order to increase the density of integration, most of the mother boards contain two nodes. The resulting 2000+ boards are interconnected in a $32 \times 34 \times 2$ mesh.

The national laboratories have both classified and unclassified supercomputer applications. Although the Paragon could run in either classified or unclassified mode, there was no way to partition the machine to allow classified and unclassified codes to run simultaneously (e.g. a classified production run on part of the machine while the remainder was available for visualization of unclassified data). To allow greater flexibility, ASCI Red is partitioned into three sections: classified, unclassified, and floating. The classified and unclassified partitions are always physically disconnected, and the floating section is connected to at most one of the other two sections. The classified and unclassified sections have full access to the service nodes and disks dedicated to their sides; these two sections are currently two of the world's most powerful supercomputers by themselves (400GF), but in this context they are relatively small. The floating section contains most of the compute nodes (approximately 1TF). In the current plans, when the floating section is added to one of these partitions, application codes have access to about 3/4 of the computing resources.

The interconnect in the ASCI Red machine consists of two types of chips. A network interface chip (or NIC) provides bidirectional direct memory access (DMA) from a node to the network at 400 MB/sec. A mesh router chip (MRC) has six ports which can be dynamically switched to create wormhole-routed [23] connections. In particular, one port is connected to a node NIC and the other five can be used to form a two-plane mesh. A header packet specifies offsets in each mesh direction (actually the routing is z,x,y,z to provide redundant routes). The MRC chips match incoming packets with outgoing ports in a fair manner. Data packets follow the path established by the header until they reach their destination. A tail packet (often a data packet with a flag set) "tears down" the path and frees channels for use by other packets. The Intel MRC chip also contains advanced provisions for error detection and correction and for the sharing of physical links among virtual lanes, which can reduce congestion and give priority to critical messages.

Although a node of ASCI Red has only small-scale (two-processor) *shared-memory processing* (SMP) capability, the overall design includes hardware assistance for remote memory access. Final status of this hardware and the software to control it has not been resolved but the intent is that pieces of the memory on most nodes or all of the memory on a few nodes can be mapped into the address space of a local node.[2] The hardware should catch load and store requests which are mapped to remote memories, provide interconnect routing, maintain cache coherence, and behave to the memory bus like a local memory bank so that the processor can proceed without waiting for a response. Software will allow maps to be set up and account for situations which are beyond the capabilities of the hardware.

The system software for ASCI Red partitions the nodes into three logical groups: service, I/O, and compute [13]. Since the service partition is relatively small (on the order of ten nodes) it runs a variant of OSF1/AD, which supplies a standard workstation image to users. In order to integrate the service partition into the machine, Sandia has added an interface module, *yod*, which allows parallel tasks to be launched into the compute partition and to be managed from the service partition. If desired, the yod module can also enable communication between a parallel application and a serial "host" node process or even processors on another machine.

Ideally, I/O partitions can be scaled with I/O needs. Since ASCI Red's I/O partition is currently of moderate size (fewer than 100 nodes), it can also use the OSF1/AD variant. Sandia and Intel have developed an I/O partition interface, *fyod*, which allows compute nodes to efficiently transfer I/O data to the I/O nodes and a parallel file system, *PFS*, which allows data to be striped across multiple disks. The I/O nodes then transfer data to and from storage.

For the compute partition, where unlimited scaling and highest performance are critical, Sandia and Intel have developed a light-weight kernel called Puma/Cougar. Puma is designed to use minimal resources; whenever possible, services are deferred to the service or I/O partition. Puma's resources are concentrated on efficient process management, memory man-

[2]One restriction is that the Intel processors have only 36 bits of addressability and the entire memory is over .5TB – thus 40 bits of addressability.

agement, and interprocessor communication. Although it supports multiprocessing, Puma is tuned for the high-performance case where a single user controls the entire node with a single process. Puma provides the convenience of a virtual memory space (i.e. to maintain the standard code linking model). However, in order to avoid the high on-node costs and the crippling communication costs of waiting for pages to be brought in from secondary storage, it does not provide demand paging. Puma uses an innovative concept called Puma portals [30, 35] to receive messages. A portal is an opening into the address space of a user application. Incoming messages are deposited directly into the location the user specified. This avoids an expensive memory-to-memory copy operation and reduces overhead significantly.

Two additional ASCI machines are expected in early 1999. They are usually referred to as *ASCI Blue Pacific* (being built by IBM for Lawrence Livermore National Laboratories) and *ASCI Blue Mountain* [3] (being built by SGI/Cray for Los Alamos National Laboratories). Both of these development efforts include initial (below-spec) systems that are currently being delivered and two planned upgrades.

As currently planned, ASCI Blue Pacific will have 512 nodes, each of which consists of 8 IBM RS-class processors. Within a node, the 8 processors will be connected by a new crossbar switch with GB/sec bandwidth and low latency. The interconnection between the nodes will build upon the SP2 technology. The OS will be a specially enhanced version of AIX. Blue Pacific will have a peak performance of 3.2TF, 2.5TB of RAM, and 75TB of disk space.

ASCI Blue Mountain will use the Origin 2000 technology to explore the use of much larger (256 or 512)-processor shared-memory components. It will also make use of Cellular-Irix, currently under development, a new operating system designed for very large SMPs. LANL has contracted for both a 3.2TF machine and a 1TF machine that will operate independently but which may be connected if needed. The memory and disk capabilities are similar to Blue Pacific.

3. The age of computational simulation. The predicted advent of GF/sec computational nodes and the ability to economically cluster multiple nodes into a single computational engine will soon enable the use of computational simulation tools by a wide range of university science departments and industrial engineering groups. The simulation of physical phenomenon, from first principles quantum dynamical systems, to micro-properties of materials, to macro-properties of fluid flow, chemical reactions, and molecular dynamics, will join experimentation and theory as mainstays of science. It will be possible to proceed directly, within a single computational framework, from a CAD design to test simulations to parametric optimizations to instructions for machine tools.

[3]The Blue distinguishes the time of purchase from the Red machine. Pacific and Mountain refer to the time zones where the machines will reside.

Though the predictions above may seem fanciful to some, they are fairly direct extrapolations from the computational experiences at large national laboratories such as Sandia National Labs. For many years these labs executed the computational simulations necessary for their national-security missions using special-purpose machines like the Cray vector series and the nCUBE hypercube machines. However, recently there has been a move toward tightly coupling standard microprocessors. The processors of the ASCI Red machine are the same processors that are available in commodity PCs. The motherboards described in Section 2 are laid out more compactly[4] than motherboards for standard PCs, where space is not a big concern, but are otherwise standard. The interconnect is special-purpose, but it still consists of only two types of chips: the NIC and MRC. Of these two, the MRC technology is currently further from the mainstream.

Despite the proprietary nature and special-purpose use of the Intel MRC chips, switching methods have a long history in the literature [6, 10] and are in use in open designs like the Myrinet system from Myricom. There is no reason to expect that reasonably-priced, PCI-based switches using these techniques will not be readily available within the next few years. In the meantime, groups at many labs and academic departments have begun building small-scale systems similar to the Sandia system by using standard ethernet and ATM interconnects [20, 32]. By the turn of the century the basic technology should be widely available to enable economical construction of department-scale machines that can perform multi-petaflop simulations over the course of a weekend (48hrs x 100GF/sec).

4. Applications at Sandia[5]. One of the biggest barriers to the use of cost-effective, distributed-memory supercomputers is the *perception* that it is very difficult to write applications codes for these machines. Counter to this perception, researchers at Sandia have discovered that typically only a small fraction of their development effort goes toward distributed memory issues – most of the effort is in adding more complex and detailed physics and chemistry to the code and assuring numerical stability and rapid convergence for the larger problem sizes that can now be tackled. One reason for this relative ease of data layout and message passing is that DOE's investment in high-performance parallel machines has been matched by a significant investment in tools that facilitate the development of parallel applications.

For example, load balancing tools like Chaco (see below) can eliminate much of the bookkeeping required to develop efficient distributed-memory software. A key to distributed-memory programming, (and for

[4]Even so, ASCI Red still requires over 70 full-sized cabinets.

[5]This section draws heavily from the Sandia-maintained web pages, accessible via www.cs.sandia.gov. These pages contain further details about the codes described in this section, pictures and videos of sample simulations, and descriptions of additional codes and tools.

all efficient programming on cache-based, RISC architectures, both serial and parallel) is the careful layout of data. Fortunately a good layout of data for cache-usage is often a good start for layout of data for parallel computing. Furthermore, a good understanding of the interplay between sections of data is important to understanding stability and convergence in simulations of physical systems. Thus an application developer can often determine manageable-sized chunks of work and the interactions between them and use Chaco to handle the messy details of data layout for efficient computation and interprocessor communication.

By following this simple strategy of decomposing problems into natural pieces, Sandia researchers have developed high-performance parallel codes for a wide variety of application areas. The dominant application areas involve simulations of physical systems such as heat transfer, chemically reacting flows, transient solid dynamics, and 3D seismic imaging. Parallel algorithms for new applications in optimization and microsimulation require very different parallel algorithmic methods. The following sections describe parallel tools developed at Sandia and describe a variety of significant applications that illustrate the types of problems that we solve with distributed-memory machines.

4.1. Parallel tools.

System Software. The use of any computer begins with its system software. The development of MAC OS and Microsoft Windows has fundamentally changed the way users interact with computers. However, high-performance computing users have continued to rely on proprietary operating systems and on variants of the venerable UNIX system. At Sandia, we have learned that it is critical to have system software which is tuned and tailored to high-performance computing.

In order to achieve the best performance from the Intel Paragon, we installed the SUNMOS operating system developed by Sandia and the University of New Mexico. SUNMOS requires less than 256KB of memory and achieved 2.5 times the communication rate of the original operating system, OSF [22]. The efficacy of the SUNMOS operating system was demonstrated in December 1994 when it was used to achieve a sustained performance of 281GF on the MP Linpack benchmark (a record at that time).

In the joint development of the ASCI Red machine, Sandia and Intel developed a second-generation operating system, Puma. The Intel version, called Cougar, enables the over 9000 processors to work efficiently together. A special communication technique, called portals [30, 35], allows applications to efficiently use the 400MB/sec links between nodes. Cougar automatically routes messages between nodes using an alternate path when necessary to avoid components which have failed or which are being upgraded. Cougar also accesses the virtual plane mechanisms which allow multiple messages to share a single wire in order to increase predictability and decrease congestion.

Puma is designed to avoid many of the costly functions and daemons of standard operating systems which provide convenience to interactive users but increase overhead for parallel computations. We are currently integrating it with Linux through a partition model [13] in which Puma can supply performance on most nodes and Linux can supply convenient features on a few nodes.

Load-balancing tools. The Chaco software package [18], developed at Sandia, minimizes communication requirements for statically load-balanced settings. When solving a set of PDEs using finite-element methods over a static mesh of an object, the computation and communication patterns repeat each iteration. A single iteration can be represented by a weighted graph where each node represents a computation, each edge represents a communication, and weights represent quantity or volume. For machines such as the Paragon or the CM-5 where the physical topology does not significantly effect the cost of sending a message, the question of balancing load while simultaneously minimizing communication can be well-approximated by the combinatorial problem of *graph separation.* The graph separation problem accepts a weighted graph as input and partitions the nodes into two groups such that the sum of the node weights is roughly balanced (for example, neither side has more than 2/3 of the total weight), and the sum of the weights of the edges going between the two groups is minimized (over all partitions satisfying the weight-balance constraint). This can be generalized to partitioning into more than two groups. If nodes of the computation are assigned to processors as suggested by such a partition, one can expect reasonable load balance (total weight of computation on a processor compared to the average weight), and minimum global communication. Although the graph separation problem is NP-complete [11], the Chaco package produces good approximations.

Dynamic load-balancing methods are currently being developed to facilitate the parallelization of methods like adaptive mesh refinement. For example, Devine and Flaherty [7] use a modified tiling procedure to guide local work exchange when using adaptive grids to solve hyperbolic systems of conservation laws, such as Burgers' equation and the Euler equations of inviscid flow.

Linear solvers. Aztec [19] is a parallel iterative library for solving linear systems, which is both easy-to-use and efficient. Simplicity is attained using the notion of a global distributed matrix. The global distributed matrix allows a user to specify pieces (different rows for different processors) of his application matrix exactly as he would in the serial setting (i.e. using a global numbering scheme). Issues such as local numbering, ghost variables[6], and messages are ignored by the user and are in-

[6]Ghost variables are copies of variables owned by other processors, but relevant to the local computation. They allow the serial code to function correctly between communications. With asynchronous communication, they may also hide latency.

stead computed by an automated transformation function. Efficiency is achieved using standard distributed-memory techniques; locally-numbered submatrices, ghost variables, and message information computed by the transformation function are maintained by each processor so that local calculations and communication of data dependencies are fast. Additionally, Aztec takes advantage of advanced partitioning techniques (Chaco) and utilizes efficient dense matrix algorithms when solving block sparse matrices. Aztec includes a variety of numerical methods including conjugate gradient (CG), conjugate gradient squared (CGS), stabilized biconjugate gradient (BiCGSTAB), generalized minimal residual (GMRES), and transpose-free quasi-minimal residual (TFQMR) as well as numerical preconditioners such as point & block Jacobi, Gauss-Seidel, least-squares polynomials, and overlapping domain decomposition using sparse LU, incomplete LU (ILU), and block incomplete LU (BILU) within domains.

Automatic meshing. The CUBIT mesh generation/grid generation environment [31] is a two- and three-dimensional finite element mesh generation tool. It is a solid-modeler-based preprocessor that robustly, and automatically (that is, with no human intervention) generates quadrilateral-element meshes for surface solids and hexahedral-element meshes for volume solids. These elements are shaped to maximize the numerical stability of the subsequent finite-element computations. For example, CUBIT avoids elements with very small (solid) angles. CUBIT currently incorporates a variety of algorithmic techniques including paving, mapping, and sweeping. Sandia is developing fundamentally new algorithmic techniques for mesh generation which will be incorporated into the package as they mature.

4.2. Parallel applications.

4.2.1. Simulations of physical systems.

Shock Physics. One of the first applications codes to take advantage of massive parallelism at Sandia was CTH. CTH is a shock-physics code designed to simulate the effects of very high-speed impacts and the resulting shock waves. While CTH typically is used within DOE to simulate weapons impacts, it can be used for any high-impact problem. For example, it was used to predict the size and shape of the impact plumes of the Shoemaker-Levy 9 comet into Jupiter and thereby aid astronomers in planning their observations. Recently it was used to simulate the impact of a comet into the Atlantic ocean. The predicted tidal wave dwarves the New York skyline and the atmospheric effects are world-wide.

CTH derives its power and flexibility from its ability to model many materials at once, track large deformations, incorporate solid mechanics principles, and does so accurately enough to follow strong shock waves. In order to solve particular problems it includes simulation models for multi-phase, elastic-viscoplastic, porous and explosive materials. For example, CTH can model concrete as it liquifies under intense shock pressure.

Numerically CTH is relatively simple. It uses three-dimensional rectangular meshes; two-dimensional rectangular, and cylindrical meshes; and one-dimensional rectilinear, cylindrical, and spherical meshes. It uses second-order accurate numerical methods to reduce dispersion and dissipation and produce accurate, efficient results. The resulting discretizations are thus relatively simple to scale to very large-sized problems.

Solid Dynamics. Recently Sandia has been developing a successor to CTH (with added capabilities) called ALEGRA. ALEGRA is a multi-material, Arbitrary-Lagrangian-Eulerian code for solid dynamics. As in CTH, ALEGRA can include complex models of the interactions of multiple materials in multiple physical phases, but it can also perform structural analysis.

ALEGRA gains much of its flexibility by incorporating multiple mesh discretizations. When an object is subject to a large external flow, the object is represented by a stationary (Eulerian) mesh and the material flows through the mesh. When it is preferable to view the object as moving through a background material, the mesh can move with the material (Lagrangian) so there is no flow between elements. For complex situations the the mesh motion can be entirely independent of the material motion (Arbitrary).

All three mesh types can coexist in the same problem. One part of the mesh can require Eulerian algorithms to model large deformation flows or penetrations while another part of the mesh requires Lagrangian algorithms to model small deformation structural response and yet another part of the mesh can have arbitrary mesh motion to follow 'near Lagrangian' flow fields. A mesh can even change type as the calculation progresses.

Typical problems solved using ALEGRA include: simulating the interactions of soil and a building during an explosion, simulating the tooling of a block of aluminum by a tungsten/steel blade, and simulating the impact to a human head during a crash.

The use of many types of meshes can make it difficult to specify a specific instance of a problem. Research is continuing at Sandia into ways to more quickly specify the geometry of a problem and to quickly create high-fidelity, three-dimensional meshes of complex objects. Some of the latest research is in the use of H-adaptivity to dynamically refine meshes only in those areas where accuracy or stability is needed. Within ALEGRA this effort is called HAMMER. The algorithms in HAMMER can currently refine the three-dimensional region of an oblique impact between two solid bars. Localizing the refinement greatly decreases the computational power necessary for the simulation.

Impact on Industry. Sandia has applied its expertise in high-performance simulation to many industrial problems. Here we describe briefly two examples, the SALVO project for siesmic imaging [25] and the the MP-SALSA project for chemically-reacting flows [29].

SALVO uses a parallel solution of the three-dimensional wave equation to image sub-surface geophysical features based on field recording of wave reflections in siesmic surveys. The partner oil companies use the images to decide whether there is likely to be oil under a particular region and if so what is the best place to drill a well. Since the wells are very expensive to drill, any increase in the expected output of a well can be quite profitable. SALVO must simulate the wave propagation through a wide range of features such as sedimentary layers and salt domes. It also includes sophisticated signal processing to dampen echos and other artifacts. Current versions of SALVO run on a wide variety of platforms including Intel Paragons, IBM SPs, and Cray T3Ds.

The MPSALSA project combines the ability to simulate fluid flows with the ability to simulate chemical reactions. The code has been a primary stimulus for the development of efficient[7] parallel iterative, sparse matrix solvers. Complex chemically reacting flow simulations are important for many critical technology areas of interest to federal agencies and to U.S. industry. These areas include: combustion research for transportation, atmospheric chemistry modeling for pollution studies, chemically-reacting flow models for analysis and control of manufacturing processes, surface catalytic reactors for methane-to-methanol conversion and chemical vapor deposition (CVD) process modeling for production of advanced semiconductor materials.

Material properties. An important component of any physical simulation is a detailed understanding of material properties. Some properties can be readily obtained from experiments but there is a growing need to calculate material properties from first principals. Codes designed to simulate materials at the molecular and quantum level provide distinct challenges for high-performance computing as opposed to the larger-scale simulation described above. Rather than attempting to create a discretization over which physical reactions can be integrated, a materials code attempts to model the individual components of a material such as atoms or electron energy levels.

The LADERA program [17] models gradient-driven diffusion through porus networks (like amorphous materials). Molecular dynamics, the simulation of the motion of individual molecules, is used throughout the simulated system volume. The techniques of Grand Canonical Monte Carlo are used to maintain two local chemical potential control volumes which control the chemical potential of each species via particle insertion and deletion. The transport diffusivity is measured by establishing a steady-state chemical potential gradient, measuring the flux and gradient of the resulting steady-state density profile, and using Fick's Law much the same way as in experimental systems.

[7]The code has twice been nominated as a finalist for the Gordon Bell Prize for advances in computational performance.

QUEST is an electronic structure code, using LDA (local density approximation) with an LCAO (linear combination of atomic orbitals) basis represented as contracted Gaussians. QUEST uses ab-initio pseudopotentials and can handle `s, p` and `d` states.

4.2.2. Optimization methods.

DAKOTA. Optimization methods are becoming increasingly important in the engineering sciences as computational resources increase to permit the use of optimization for automated design. Engineering-design problems typically utilize physical simulation(s) to evaluate the utility of a given design. Consequently, the use of optimization methods to perform a systematic search of feasible design parameters requires the computation of many simulations.

The DAKOTA software toolkit [9] incorporates a variety of optimization methods that use parallelism in one of two ways: by performing multiple evaluations of the objective function simultaneously for multiple parameter settings or by parallelizing the objective function evaluation itself for a single set of parameters. The first approach is useful if each simulation is not too expensive; it performs simulations independently on each processor in a master-slave fashion. This approach is useful, for example, for optimization methods that compute finite differences to estimate the gradient at a point, which involves the simultaneous evaluation of a group of design parameters. The second approach is useful when each simulation is expensive; the parallelization of the simulation serves to reduce the total time of execution. Efforts are currently underway at Sandia to develop a capability within DAKOTA to combine both of these capabilities, enabling parallel simulations to be run independently on a collection of processors by a master optimization process.

Heuristic Global Optimization. The SGOPT global optimization library [15] provides a common interface to parallel-optimization methods. The parallel global search can be guided by a single master process or distributed among many coordinated processes. Users will ultimately be able to quickly apply, for example, a parallel simulated-annealing method to a new application by specifying an objective function and search neighborhoods.

The current version of SGOPT includes a variety of parallel genetic algorithms. Genetic algorithms are a general heuristic search technique. Each algorithm maintains a *population* of candidate solutions, which can be ranked by a scoring function. New generations of candidate solutions "evolve" from the previous population using specified *mutation* operations applied to individual solutions and *crossover* operations, which produce a new solution from two previous ones. The best candidates survive to produce the next generation. Parallel genetic algorithms typically create new candidates locally and coordinate the global search by communicating the best solutions seen on each processor; the communication topology is typically sparse and load balancing is generally not an issue for these algorithms.

Achieving high parallel performance for distributed global optimization algorithms requires an algorithmic investigation into the effects of parallelization. For example, we have observed that asynchronous communication appears to improve the search performed by genetic algorithms in addition to eliminating the idleness introduced by synchronization [16]. That is, communicating *too* often in genetic algorithms leads to premature convergence to solutions that are generally poorer than those found by more loosely synchronized implementations.

Branch and Bound. Sandia has recently begun developing PICO, a library of Parallel algorithms for Integer and Combinatorial Optimization whose initial focus is branch and bound. Branch and bound is an exhaustive search technique which proceeds by recursively dividing the feasible region into subproblems and discarding suboptimal subregions (where the bounding procedure returns a value worse than the best feasible solution found so far). Each subregion corresponds to a node on a search tree. In PICO, nodes are processed in parallel asynchronously. PICO, run on ASCI Red, should be able to solve much larger instances of combinatorial problems than previously possible because of the huge number of processors and large memory size (hence the ability to search larger trees).

As with parallel global optimization, the role of communication plays a critical role in parallel branch-and-bound algorithms. For example, parallel branch-and-bound algorithms can exhibit slow-down effects when the global search is not well coordinated, thereby allowing some processors to unproductively explore the branching tree [8]. PICO uses one or more master processors to control the selection of search nodes to be processed [8]. Processors must coordinate a variety of communication tasks, including: (1) communicate to distribute the load (make sure that all available processors are working on subproblems if possible), (2) distribute the *quality* of work (make sure that processors are not searching nodes that are likely to be suboptimal when other, better nodes, are not being processed), and (3) propagate the best solution found so far (so that some subproblems can be determined suboptimal and dropped).

5. Using department-scale parallel machines.

In the previous sections we have described how very-large systems have been built at Sandia and used to enable a new simulation capability. We have argued that it will soon be possible for groups of researchers, such as academic science/engineering departments and industrial design teams, to afford similar capability. In this section we discuss issues related to making these computational capabilities readily usable by all potential users. In many cases these issues are extensions of concerns that many users have about Unix workstations and servers. The potentially heterogeneous nature of department-scale machines, the distributed nature of their architectures, and the desire to reach a wide class of users magnifies the problems and in many cases necessitates new research directions.

This section divides the issues into the following pieces: usage models, programming models, resource management, data movement, system reliability, and code evolution.

5.1. Usage models. The usage model for most supercomputers is fairly simple. Parallel programs are compiled serially on a portion of the supercomputer or on a cross-compilation platform. Special libraries to enable use of the particular supercomputer are linked with the compiled code. Standard input and output are funnelled through a proxy process and other output can be directed to disk or tape resources. The usage model can depend upon scheduling and resource-management strategies which are discussed in Section 5.3.

Unfortunately, even this simple model can vary considerably in detail from machine to machine. Users must know where to find the correct versions of compilers and libraries. Disk space is often limited on the supercomputer so executables may have to be copied onto the machine for each use. "Standard" libraries such as BLAS and MPI are rarely identical in behavior (eg. BLAS routines can have different calling conventions and MPI routines can have different buffering strategies). The method of launching the application tends to use a proprietary command sequence with wildly varying conventions for specifying the parameters of the run such as the number and type of nodes, the location of executables, the location of disk files, etc. Debuggers and performance monitors, when they exist, can use interfaces which differ from the launch interface.

The reader who is familiar with Unix workstations will recognize that many of these issues must be faced when creating code that is portable across multiple vendors' workstations, or multiple generations of workstations from a single vendor. For the department-scale supercomputers envisioned here these problems compound, since the supercomputers may contain components from multiple vendors and multiple generations which must be used simultaneously. The user community should include scientists and engineers, many of whom are not familiar with Unix (and who don't want to be). Some method of simplifying interaction with the system will be necessary in order enable the routine use of these machines by individuals who are not experts in parallel computing.

Since, it is unlikely that any one particular set of interfaces and libraries will replace all others, we envision the creation of a flexible machine-independent front-end interface. This front-end might follow the current trend toward web-based forms. Users can choose common parameters such as machine, number of nodes, or name of executable from pull-down menus. Machine-specific, site-specific, and user-specific profiles could then fill in many attributes such as locations of compilers, libraries, temporary files, and launch sequences. The front-end could translate this to the local machine syntax, warn the user when certain features were not specified or were not available, and launch the application and/or create scripts for faster future compilation and invocation.

Standardizing interfaces to debugging and performance monitoring tools may prove more difficult because the requirements for these tools continue to change. Standard text-window approaches quickly become unwieldy as the number of threads of parallelism expand. Consequently, tasks such as monitoring progress, debugging, and performance tuning will require new tools. One approach is to use advanced graphical and virtual reality techniques to encode information about all threads so that the user can navigate through the data. It remains to be seen how well these approaches will scale on very large machines, but they have promise for department-scale machines. Data filtration methods will evolve as we learn more about how human's can and do integrate information. Such tools will be helpful on small systems, but critical for large ones.

5.2. Programming models. A programming model defines the way an application interacts with the system to access resources such as memory, I/O, and processors. For example, the programming model for the Intel Paragon and ASCI Red is a distributed-memory model with statically allocated processors; message cost is modeled by a fixed start-up cost and, for large messages, an additional length-dependent cost, but does not depend upon which pair of processors is communicating.

We believe this programming model is a good starting point for department-scale machines. Although shared-memory models offer a seemingly simpler programming model for parallel machines, they are unlikely to resolve difficulties associated with the development of high-performance parallel algorithms. Shared-memory models obscure the fact that each processing element sees a hierarchy of access time for data. As machines grow, the effect of this hierarchy becomes more pronounced. Increasing the number of processors by an order of magnitude will increase the number of levels of the hierarchy and/or increase the worst-case gap between levels. Consequently, parallel software using a shared-memory model will probably not perform as well as one based on a distributed-memory model, where data movement is explicitly determined.

Several standards have recently emerged that begin to provide a more uniform programming model for distributed-memory parallel machines. Standards like MPI [14] and POSIX threads [27] provide general capabilities for communication and coordination of computation respectively that were previously available only in vendor-specific libraries.

Threads allow a single processor to multitask without the overhead of process swapping. Complex parallel codes, such as parallel optimization, can be conveniently partitioned into multiple independent (parallel) tasks. Ideally, the user should be able to allocate these tasks to processors arbitrarily, using threads to provide modularity. To achieve this, each thread must be a *communicator* within MPI.

Although we have argued that pure shared-memory programming is not ideal for large simulation applications, computing components will increasingly include symmetric multiprocessors. SMP components that contain 4-8 processors are currently available (e.g. SGI O2000, DEC 8400, SUN Starfire), and will increasingly be used as base components for larger clusters. As described in Section 2, the ASCI Blue machines will have this basic design, ultimately with much bigger SMP nodes.

Programming models for stand-alone SMP machines have been strongly influenced by database applications. Since no machine of this sort has been built at either the ASCI scale or department scale, researchers and vendors must develop programming models for general-purpose scientific computing on these types of machines. It's possible that no single model may ultimately suffice for all users depending upon the size of the SMP nodes, the sophistication of the user, and the desire for performance vs. quick development. For example, a simple programming model might treat each physical processor as a separate node. This model provides good continuity with current simulation work; supporting this model would enable simulation codes to run seamlessly on a cluster of SMPs with some performance penalty. Alternatively, one could augment the memory hierarchy of this simple model by one level to recognize that off-processor communication with any member of the same SMP node is faster than communication with any more remote processor. A more sophisticated programming model might explicitly reflect the physical characteristics of the machine, such as the topology, to provide better data movement.

Finally, the manner in which processors are allocated and programs are executed is an important component of the programming model. The ASCI Red machine provides static allocation of processors on which a program is run in a SPMD fashion – a single program is loaded onto a statically-defined number of processors. Each copy of the program then processes its own data and communicates with the other copies via some interface. Both PVM and the MPI2 standard extend the model to allow applications to request and colonize additional processors during execution. In this model an application might start on a single processor and, based on the input, request groups of nodes to be used for specific subtasks. During execution the application can even release nodes which are no longer needed.

The implications of extending the parallelism model beyond static allocation are significant – particularly for the system software. Resolving competing requests for resources, supplying consistent names, and mediating communication are just a few of the hard questions to be resolved. However, there is a great incentive to solve these problems since their resolution will also be useful in creating systems which can dynamically adjust to faulty hardware. It is hard to predict what breakthroughs will enable more flexible parallelism. In the short run, parallelism will probably be mostly static. Simple forms of dynamic growth such as pre-allocated nodes but dynamic allocation of processes (e. g. allowing a process to allocate from a fixed-size "heap" of processors) and pre-defined servers to which applications can *attach* will then allow experimentation to determine the true needs for dynamic growth. Operating systems will also have to resolve some of the naming and code-migration issue to allow (software) swapping of a spare physical processor for a faulty processor within an otherwise statically-allocated environment.

5.3. Resource management. The departmental supercomputer, by its very nature, will be a shared and valuable resource. Some method will have to be developed to adjudicate between various types and sizes of usage requests. In this section we discuss fundamental resource-allocation choices such as interactive vs. batch scheduling, time-slicing vs. dedicated computation, and static vs. dynamic processor allocation. We also consider scheduling issues for different management policies, fragmentation, and performance evaluation.

In an interactive environment, the user submits a job to be run "while you wait." If the resources are not available, the job is denied. In a batch system, a request is queued if necessary and run when the resources are available according to a site-specific priority policy.

A common technique for minimizing denial in interactive settings is to *time slice* the use of processors. Time slicing allows many requests to receive small amounts of service which it is hoped are enough to meet the users' immediate needs. Network-of-workstation systems, PVM systems, and IBM SPs typically allow processor-level time sharing via the standard Unix process interfaces. This is in contrast to *dedicated* systems where each processor runs at most one (perhaps multithreaded) process at a time.

The ASCI Red machine uses NQS to provide static allocation of dedicated processors with both interactive and batch scheduling. During the day, the processors are partitioned into an interactive section and a short-term batch section. At night the entire system is dedicated to large batch jobs. A job must specify the number of processors it requires and an estimate of the time it will run. In the interactive section, a job is run immediately if the processors are available. Otherwise, it is denied. For the batched queues, jobs are submitted with a priority which grows as the job waits in the queue. The highest-priority job that can fit on the number of available processors is run, unless the globally highest-priority job has crossed a priority threshold. At this point, released resources are held idle for this job until enough are available for it to run.

There is no requirement that processors allocated to a particular process be physically close to each other, though this is done where possible. Such a policy could lead to bad fragmentation. However, the machine is cleared of all jobs twice per day when switched between day and night policies. In practice this keeps fragmentation to an acceptable level. There is some underutilization of resources just before switching to daytime mode, since there are few jobs in the queue short enough or small enough to use the "leftovers" from the night's big runs, and no interactive users allowed.

We believe this basic division of resources is a good starting point for department-scale simulation engines. In particular, we believe that dedicated processing is more appropriate than time-slicing. First, simulations tend to require large amounts of memory. Time-slicing mechanisms rely on the ability to quickly switch between users and this is not possible if each user wants most or all of the memory. Second, time slicing in parallel applications can lead to huge inefficiencies. If one portion of the application must wait for another which is swapped out then resources can be idled and deadlock is possible. While "gang-scheduling" processes can eliminate deadlock by ensuring that all processes of an application run at the same time, it is probably not the most appropriate use of the resources. The best use of parallel hardware is achieved by devoting the vast majority of the processors to servicing the large simulation jobs which require the computational and/or memory resources of the full machine. Since applications that *require* time slicing do not fully utilize a tightly-coupled machine, we believe that they can best be performed on clusters of workstations or on standard Unix servers.[8]

An additional benefit of dedicated use is simplied performance evaluation. It is much easier to compare the performance of competing algorithms when they are run in dedicated mode. Otherwise, one must determine how long each processor ran the application, calculate which portion of message latency was caused by swapping, ensure that swapping does not *hide* latency, and so on.

We have already argued that most resources should be dedicated to large production runs. These are stable codes which can run without operator intervention and can direct output to files. Therefore, batch scheduling is appropriate for these runs. However, some smaller portion of the resources must be available for interactive use for code development. Interactive use not only allows immediate feedback, when resources are available, but also allows interactive debugging, visualization, and human guidance for parameter tuning. A university department, where some subset of researchers is active at any given time, will probably need to have interactive capability 24 hours a day most of the time, with special arrangements made for runs requiring the full machine.

Regardless of basic resource allocation policy, explicit scheduling of queued tasks is difficult and will require policies more flexible than NQS in order to achieve acceptable performance as the complexity of resource demands increases. There are not only multiple processors, but also multiple auxiliary resources such as disks and/or tape. Without preemption, long jobs requiring many resources can effectively starve many small jobs.

[8]Tying high-performance servers with workstation-level servers into a single computational resource would force the reevaluation of these issues, but such systems are beyond the scope of this paper.

However, a steady stream of small jobs can also lock out a large job if they are continually scheduled into small groups of newly-released processors. At Sandia, task queues tend to be long, so there are many choices for the next job to run. Furthermore, time estimates should be treated only as an upper bound. Jobs can be terminated if they exceed their time estimate, but such a policy encourages users to inflate their running-time estimates. The development of more flexible queueing systems is already underway including LSF and IBM's loadleveler.

5.4. Data movement. We have argued that the natural progression of technology will enable tightly-coupled systems consisting of a hundred powerful PC-class nodes by the end of the century. There will be, however, a significant gap between the speed at which memory internal to a node and external to a node can be accessed. Keeping this gap as small as possible and/or mitigating it in software as much as possible will be a key to making the described systems general enough to handle the wide class of simulation problems for which they are constructed.

Computer vendors, national labs, and universities have all been working hard to reduce the cost of moving data between processors. One approach has been to add hardware to manage data movement asynchronously to main computation. Sometimes the hardware is used to manage protocol stacks. For example the IBM SP series has smart channels and an attached 386 processor, and the Intel machines have a message co-processor mode in which one processor within an SMP node handles only communication. Other hardware is designed to move data directly from memories to the network without interrupting the processors. For example, Intel allows block transfer commands to be loaded to a direct-memory-access (DMA) engine, and Cray has implemented a variety of block transfer engines and special data-movement registers.

A second approach has been to attack the software overhead. Sandia has implemented a new protocol called *portals* which allows data to be moved directly to and from user-level addresses, thereby avoiding memory copies. Additionally, Chien [26] has developed fast messages and a variety of groups are working on active messages [34]. All of these approaches concentrate on allowing the data to access hardware as soon as possible. They take advantage of the fact that system-area networks are much more reliable than the typical LAN, WAN, or internet connection.

A system which provides general-purpose communication with 10μs latency is currently considered very good. However, some special-purpose systems can operate in the 1μs range. Reduction much beyond 1μs may be difficult for large machines based purely on speed-of-light constraints, but 1μs latencies seem a reasonable goal.

In order to achieve 1μs latencies, it will probably be necessary to harness some of the hardware currently used to mask local memory latency. For Ghz processors, nominally 40ns memories, and nominally 1μs network latencies, the ratio of cache memory speed to local memory to remote memory is 1:40:1000. Thus, the gap between remote and local memory may be smaller than the gap between local memory and high-speed cache. Advances in multi-chip packaging and memory technology may shift this ratio slightly but it is likely to be about equal for the foreseeable future. Thus it seems appropriate to look at the techniques designed to mask the cache-to-processor-memory gap.

Latency-masking techniques can be divided into three classes: fetching data early, keeping data near-by for reuse, and doing something else while waiting for data to arrive. Two common ways of fetching data early are to issue explicit pre-fetch commands and to bring in entire cache lines under the assumption that wanting one word means the others will be needed soon. Caches are of course the paradigmatic method of keeping data nearby. Pipelining, multi-threading, and out-of-order execution are a few of the ways in which useful work can be performed while awaiting data. For writes there is also the possibility of not waiting for data to finish being moved before continuing with computation.

All of these mechanisms typically make use of a combination of special hardware and compiler assistance. When necessary, additional levels have been created such as floating-point registers, vector registers, write-back buffers and DRAM page-mode optimizations. In order to allow flexible movement between levels, a variety of mechanisms have been introduced to keep track of what data is where. Caches have tags, virtual memory pages are mapped to real pages etc. In fact, the mechanisms for mapping between levels have accrued caches independent of the data caches, such as the translation look-aside buffers.

The challenge for the architecture community is to understand how best to expand these local-memory techniques to remote memories, in particular how to handle the timing and reliability issues of off-node communication without unduly sacrificing local performance. For example, even beyond added latency issues, remote memory accesses can fail, and even if successful usually require acknowledgements. Current techniques tend to gather remote data into buffers which are then transmitted in large packets to amortize protocol overhead.

5.5. System reliability. The affordable systems envisioned in Section 3 will not be practical unless they can run for days or weeks without interruption. Much of the work in system reliability has assumed that pieces are loosely coupled. For example, a distributed system such as a cluster of workstations uses the TCP/IP protocol to decouple workstations. Runtime systems such as Linda [3] or Cilk [4] can then provide a very robust system via the bag-of-tasks model [2]. Many large-scale simulation codes have tasks which make repeated passes over huge data structures. These tasks are not appropriate for bag-of-tasks because frequent rescheduling will destroy data locality. This motivates our emphasis on much more tightly coupled systems. A robust tightly-coupled system must recognize and isolate hardware faults; failed components cannot bring down the whole system and they must be replaceable while the system continues to run.

In fact, one of the new design requirements of the ASCI Red machine was that the system mean-time-to-failure (MTF) be over a month even though component MTF is closer to two days. From a hardware point of view this meant that many system resources such as clocks and power supplies had to have redundant spares. In addition, the network had to be designed to survive the electrical disconnect of pieces. Thus a board containing a subpiece of the mesh can be physically removed and replaced without causing catastrophic signaling errors.

This hot-swapability is complimented by a two-plane mesh design and a redundant routing scheme that maintains routable connectivity between all healthy components even when one component is faulty. A separate diagnostic network is included to monitor the health of other components. The diagnostic hardware can isolate faulty components and initiate the rerouting and application-recovery operations.

From a software point of view, it was necessary to create an operating system that could withstand pieces being removed and replaced. The standard daemon schemes tended to propagate faults into system failures. The Puma/Cougar light-weight kernel system acts independently on each node and interacts only through tightly-coupled data-movement protocols. The OS can receive and propagate signals from the diagnostic hardware to applications. The OS is also responsible for automatically routing around failed components. It can create new routes using the flexibility provided by the second plane. In addition it can reconfigure a spare node to have the logical identity of a failed node.

With this type of system reliability, applications that are designed with a distributed-system scheme such as bag-of-tasks can continue without the failed component and even be given a replacement component. However, applications that are less fault-tolerant (such as most simulation codes) must be restarted from scratch on healthy hardware. To overcome this deficiency, we believe that parallel OSs must manage the checkpointing and recovery of applications which span faulty hardware. It is unlikely that users will develop applications that are robust to system failures since this would add significant complexity to their software development. Consequently, checkpointing technologies seem the most promising approach to ensuring that long simulations can be performed in spite of system failures. Although checkpointing is not a new idea, there remain many fundamental questions concerning the most appropriate and useful means for checkpointing parallel applications on large machines. For example, assuming that processor memory is needed for computation, checkpoint data must be stored on the external file system. Stopping a computation to checkpoint, e.g. write out all memory, can be unacceptably slow. However, checkpointing pieces in "background" mode leads to issues of data consistency and can interfere with performance evaluation.

Another problematic area for system reliability is disk and network I/O. While RAID devices increase the reliability of disks, they do not scale to the levels expected of our systems. Applying the highest level of redundancy techniques to all components will probably prevent the systems from achieving the performance for which they will be built. It will be particularly important to identify which sections require redundancy and which require highest performance, and therefore only bare-bones reliability measures.

5.6. Code evolution. Computer hardware is changing so rapidly that no one should expect a system to stay static for more than a few years. However, if scientists and engineers are going to use these systems as part of their routine work, they cannot be asked to start over from scratch every year or two. What we need is the confidence that a C or Fortran programmer has today that his or her program will run on whatever workstation comes along. The simulation code developer needs to know that his code will run on the new parallel machine, even though maximizing performance for the new architecture could be a time-consuming process. Of all the potential barriers to creation of a vibrant, affordable, simulation capability the need for a stable base is probably the most worrisome.

Standards such as MPI and POSIX threads, if and when they become truly standard, will help in this regard, since any new machine will be expected to support these capabilities. Availability of department-scale supercomputers will accelerate standardization, at least to a small number of competing options.

6. Conclusion. The national labs are in the process of developing machines and algorithms which enable simulation to complement experiment and theory in the understanding of complex 3D phenomena. Simultaneously, market forces are creating the possibility of constructing affordable, department-scale supercomputers. We envision a day soon when virtually every scientist and engineer will rely on access to simulations running on supercomputers. Although several challenges remain to be met, we are optimistic that the new millenium will usher in this new age of simulation.

Acknowledgements. We thank all the members of Sandia National Laboratories Massively Parallel Computing Research Laboratory (MPCRL), who have shared their extensive experience, papers, and web pages with us. We thank Karen Devine, Bruce Hendrickson and Jim Tomkins of Sandia National Laboratories for helpful discussions.

REFERENCES

[1] Currently the best information on ASCI can be found via http://www.llnl.gov/asci/ .

[2] D.E. Bakken and R.D. Schlichting, "Tolerating failures in the bag-of-tasks programming paradigm", *Proceedings of the 21st International Symposium on Fault-Tolerant Computing*, pp. 248–255, 1991.

[3] N. Carriero and D. Gelernter, "Linda in context", *Communications of the ACM*, vol. 32, No. 4, April 1989, pp. 444–458.

[4] C.F. Joerg, "The Cilk System for Parallel Multithreaded Computing", PhD Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, January 1996.

[5] There is no paper or tech report that we are aware of devoted to the architecture of the Cray T3D, but the following web site is excellent: http://www.cray.com/PUBLIC/product-info/mpp/T3D_overview.html .

[6] W.J. Dally and C.L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks", *IEEE Transactions on Computers*, Vol. 36, 1987, pp. 547–553.

[7] K.D. Devine and J.E. Flaherty, "Parallel adaptive *hp*-refinement techniques for conservation laws", *Applied Numerical Mathematics*, Vol. 20, 1996, pp. 367–386.

[8] J. Eckstein, "Parallel Branch-and-Bound Methods for Mixed-Integer Programming on the CM-5", *SIAM Journal on Optimization*, Vol. 4, No. 4, pp. 794–814, 1994.

[9] M.S. Eldred, W.E. Hart, W.J. Bohnhoff, V.J. Romero, S.A. Hutchinson, and A.G. Salinger, "Utilizing Object-Oriented Design to Build Advanced Optimization Strategies with Generic Implementation", paper AIAA-96-4164 in *Proceedings of the 6th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Bellevue, WA, Sept. 4–6, 1996, pp. 1568–1582.

[10] C.M. Flaig, "VLSI Mesh Routing System", California Institute of Technology Technical Report 5241, May, 1987.

[11] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, New York, 1979.

[12] D.K. Gartling and R.E. Hogan, "COYOTE - A Finite Element Computer Program for Nonlinear Heat Conduction Problems Part II - User's Manual", Sandia National Laboratories Technical Report SAND94-1179, 1994.

[13] D.S. Greenberg, R. Brightwell, L. Fisk, A.B. Maccabe, and R. Riesen, "A system software architecture for high-end computing", *Proceedings of SC97*, November 1997, to appear.

[14] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable parallel programming with the Message-Passing Interface*, MIT Press, Cambridge, MA, 1995.

[15] W.E. Hart, "The SGOPT Optimization Library", http://www.cs.sandia.gov/~wehart/proj/sgopt.html .

[16] W.E. Hart, S. Baden, R.K Belew, and S. Kohn, "Analysis of the Numerical Effects of Parallelism on a Parallel Genetic Algorithm", *Proceedings of 10th International Parallel Processing Symposium*, 1996, pp. 606–612.

[17] G.S. Heffelfinger and F. van Swol, "Diffusion in Lennard-Jones fluids using dual control volume grand canonical molecular dynamics simulation (DCV-GCMD)", *J. Chem. Phys.*, Vol. 100, No. 10, May, 1994, pp. 7548–7552.

[18] B. Hendrickson and R. Leland, "The Chaco User's Guide: Version 2.0.", Technical Report SAND94-2692, Sandia National Laboratories, 1994.

[19] S.A. Hutchinson, J.N. Shadid, R.S. Tuminaro, "Aztec User's Guide", Technical Report Sand95-1559, Sandia National Laboratories, October 1995.

[20] There are several papers on this topic in the proceedings of the Intel Supercomputer Users' Group '97 Conference (http://www.cs.sandia.gov/ISUG97/program.html) and Pentium Pro Clustering Workshop (http://www.scl.ameslab.gov/workshops/).

[21] C.E. Leiserson, Z.S. Abuhamdeh, D.C. Douglas, C.R. Feynman, M.N. Ganmukhi, J.V. Hill, D. Hillis, "The network architecture of the Connection Machine CM-5", *Journal of Parallel and Distributed Computing*, Vol. 33, No. 2, pp. 145–156, 1996.

[22] A.B. Maccabe, K.S. McCurley, R. Riesen and S.R. Wheat, "SUNMOS for the Intel Paragon: A Brief User's Guide", *Proceedings of the Intel Supercomputer Users' Group, 1994 Annual North America Users' Conference*, June 1994, pp. 245–251.

[23] L.M. Ni and P. K. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks", *IEEE Computer*, February 1993, pp. 62–76.

[24] See www.esd.ornl.gov/facilities/beowulf.

[25] C.C. Ober, R.A. Oldfield, D.E. Womble, and J.P. VanDyke, "Seismic Imaging on Massively Parallel Computers," Technical Report SAND96-1112, Sandia National Laboratories, May 1996.

[26] S. Pakin, M. Lauria, and A. Chien, "High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet", *Supercomputing '95*, San Diego, California.

[27] "POSIX system application program interface: Threads extension [C language] POSIX 1003.4a draft 8", Available from the IEEE Standards Department.

[28] "P/THERMAL User Manual", Release 2.6, PDA Engineering, PATRAN Division, Costa Msa, CA, March 1993.

[29] A. Salinger, K. Devine, G. Hennigan, H. Moffat, S. Hutchinson, and J. Shadid, "MPSalsa: a finite element computer program for reacting flow problems part 2 - user's guide", Technical Report SAND96–2331, Sandia National Laboratories, 1996.

[30] L. Shuler, R. Riesen, C. Jong, D. van Dresser, A. B. Maccabe, L. A. Fisk and T. M. Stallcup, "The Puma Operating System for Massively Parallel Computers", *Proceedings of the Intel Supercomputer Users' Group 1995, Annual North America Users' Conference*, June 1995.

[31] G. D. Sjaardeam, et. al., "CUBIT Mesh Generation Environment, Volume 2: Developers Manual", Technical Report SAND94-1101, Sandia National Laboratories, 1994.

[32] T. Sterling, D. Becker, D. Savarese, et al., "BEOWULF: A Parallel Workstation for Scientific Computation", *Proceedings of the 1995 International Conference on Parallel Processing (ICPP)*, August 1995, Vol. 1, pp. 11–14.

[33] Follow links from the following *Official U.S. Government Source* web page (as of March 1997): http://www.state.gov/www/global/arms/ .

[34] T. von Eicken, D.E. Culler, S.C. Goldstein, and K.E. Schauser, "Active messages: A mechanism for integrated communication and computation," *Proceedings of the 19th Int'l Symp. on Computer Architecture*, May 1992, Gold Coast, Australia.

[35] S.R. Wheat, A.B. Maccabe, R. Riesen, D.W. van Dresser, and T.M. Stallcup, "PUMA: An Operating System for Massively Parallel Systems", *Scientific Programming*, Vol. 3, 1994, pp. 275–288.