DOE/ER/25052--3

DE92 003456

ULTRACOMPUTER RESEARCH PROJECT

Progress Report

for Period January 1, 1991 - December 31, 1991

Allan Gottlieb, Principal Investigator

New York University
Courant Institute of Mathematical Sciences
251 Mercer Street
New York, N.Y. 10012

October 1991

Prepared for

THE U.S. DEPARTMENT OF ENERGY
GRANT DE-FG02-88ER25052

Overview

The NYU Ultracomputer project continues to pioneer the study of architecture and software for large-scale, shared-memory parallel computers. During this past year, we have achieved several very significant milestones, most notably we fabricated and used the first-ever combining switches and we increased our industrial involvement. Other important accomplishments include porting our Symunix operating system to the Ultra III prototypes; further developing a very high quality, portable C compiler (GCC version 2), needed for our prototypes, that has attracted considerable commercial attention; producing a fast solver for Laplace's equation on multiply connected domains; and furthering the analysis of buffered interconnection networks and parallel random number generators.

In addition to further developments in the areas mentioned above, we plan two new activities for next year. First, we will obtain extensive measurements of the effect of combining on scientific and other application software using both the Ultra III hardware prototypes and a new simulation environment that we are presently constructing. Our successful VLSI development of combining switches has already demonstrated that the additional cost for combining (measured in interchip wires) is about 100% using modest packaging (208 pins), 50% with 300 pins, and zero given next generation densities and 400 pins. (Note that it will continue to be the wires and not the gates that dominate the cost of an interconnection network.) Hopefully, these chips will refute the often-quoted claim that combining increases the cost of the network by a factor of between 6 and 30.

Second, we will port our operating system to the new NCR series of Intel 486-based multiprocessors. NCR has agreed to donate a machine during the first quarter of 92 for this effort, which will strengthen the ties between our project and NCR.

More detailed descriptions of our recent contributions and future plans are given in subsequent sections.

Industrial Contacts

During 1991, we have continued our relationships with AMD (Advanced Micro Devices) and MCC (Microelectronics and Computer Technology Corporation). Due to the ongoing relationship between NCR and Allan Gottlieb, the principal investigator, NCR has become interested in our project and will be contributing substantially next year. As discussed below, we have also become involved with many other commercial organizations.

The principal investigator is a charter member of two NCR committees. The Parallel Processing Academic Advisory Council assists NCR in the development of their large-scale parallel processors; Gottlieb has worked specifically with the architecture and hardware development team. The Science Advisory Committee visits various NCR plants, reviewing their operation for upper management, and provides technical and other recommendations for improved operation.

As a result of these contacts, NCR has offered to fabricate higher density versions of our combining switches within the next year. The technology used will most likely be 1-micron CMOS in 208-pin packages. In addition, NCR will provide one of their new series of multiprocessors in the first quarter of 1992. We will port our Symunix operating system to this new host.

We have had extensive contacts with many members of the 29000 development organization within AMD. The company continues to support our work financially and technically. We have received a grant from AMD for our GCC work. They have contributed all the microprocessors we have required, most recently the new 29050s, and several in-circuit emulators. In addition, they will provide at no charge the 29000 series processors needed for a 256-processor Ultracomputer. MCC plans to join us in a proposal, probably to DARPA, to construct that machine.

Our GCC compiler development has attracted considerable commercial attention. We have had frequent technical contacts with compiler developers at Shell Oil, Data General, IBM, SRC, NeXT, Dell, NCD, and Cygnus as well as with numerous academic compiler developers world wide. To support our GCC effort, IBM is placing an RS/6000 workstation in our laboratory and Data General has sent an AViiON multiprocessor containing two Motorola MC88000s. (GCC is *the* compiler shipped with AViiON and NeXT workstations;

NeXT is considering supplying a workstation for our use). Finally, Hewlett-Packard has given us a model 9000/720 "Snake" workstation, an X-terminal, and a high-speed laser printer to support our compiler work.

BBN has provided us with time on one of their Butterfly multiprocessors so that we could evaluate the performance of parallel algorithms. In addition, we have had numerous technical discussions with their OS group.

We have also spoken with Astronautics concerning the use of ultra-like networks in packet routers and with Bell Laboratories on the same issue and on VLSI design.

VLSI Research

During the present reporting period, work continued towards the goal of constructing a VLSI network that combines requests directed at the same memory location. We expect to have a sample combining network running in early 1992. Combining chips themselves are available now and have been extensively tested, both with an IMS VLSI tester and in a running system. Decombining chips have been submitted to MOSIS after having been simulated extensively. The hardware needed to test our VLSI chips and to construct Ultra III is supported by NSF.

Recent Accomplishments

Using MOSIS 132-pin packages, a 2×2 switch node is composed of four each of two types of chips: *forward path* and *return path* components. We fabricated our first version of the forward path combining chip in early 1990 and have tested it extensively. Initial testing was done in an IMS tester and verified the full functionality of the chip at speeds of up to 15MHz.

Subsequently, we built a single-board, 2-input/2-output non-combining network composed of eight of these chips with their "don't combine" signal held asserted. This board has been rigorously tested in 2-PE/2-MM systems, using both our old MC68010-based PEs and our new Am29000-based PEs. It functions correctly at all speeds at which the memory and processors work reliably (up to 15MHz) and is now in routine use in a 2-processor Ultra III prototype.

The return path component design was recently completed and submitted to MOSIS, which expects to deliver the chips in mid-November. At that time we will initiate testing in the IMS tester and construct a single-board, 2-input/2-output combining network.

To ensure that the return path components function correctly in the system, we adopted a sophisticated simulation methodology. Simulations were done at three levels. At the highest level, we wrote a behavioral simulation specifying how each switch component should perform. Next was a structural level simulator containing a register transfer language description in C for each cell in the design. At the lowest level we used a switch-level simulation of the circuit extracted from the VLSI layout.

We simulated a 16-PE/16-MM system with a 16×16 combining network (the size of the full Ultra III prototype). This simulation ran both the behavioral and structural models of the switch simultaneously and compared their results. A random stimulus was given to the system to verify expected operation. We simulated only a single component at the switch-level since running the switch-level simulation on a full network would be prohibitively expensive.

To assist in the layout verification process, gate-level schematics of each cell were done using DASH, a commercial schematic-capture system. A series of small programs were written that allowed extraction of transistor connections from both the schematic and lay-out in identical format. The resulting text files were then compared to validate each layout; a correct layout would produce identical files.

The development of these simulation and comparison techniques resulted in a longer development cycle for the return path component than we originally expected, but we now have an extremely high level of confidence that the part will work correctly in the system. This simulation infrastructure represents an investment for greater efficiency in our future design work as noted below.

While developing the return path component, we discovered a generic problem in our use of the NORA VLSI methodology that affects the forward path components previously fabricated (it makes them more sensitive to noise, which sets relatively tight requirements on clock waveforms), though these components currently are performing without error. We plan to submit a revised forward path component to MOSIS in October.

Finally, we modified the Magic technology files and CIF generation code to allow the construction of geometry that meets the design rules of NCR's state-of-the-art CMOS process. This included developing, in consultation with Bell Laboratories, a "gap-filling" algorithm to ensure that generated layers, such as wells, meet all applicable design rules. We expect NCR to fabricate chips for us during the next year.

Future Plans

VLSI research in the upcoming year will be concentrated in two areas. First, we will extensively test the return path chips that are due back from MOSIS in late 1991. In addition to using our IMS tester, we plan to construct first a 2×2 and later a 4×4 combining network for use in our Ultra III prototype (a full 16×16 network will be constructed once the smaller networks have been verified to be working correctly).

The second major area will involve making extensive use of the simulation technology developed this past year to investigate the alternative combining structures presented in our original proposal, as well as a new, more complex structure that can process 4 input requests simultaneously. Due to the validation procedure we have developed this year, it will be easy to generate a correct layout for the basic cells of any promising scheme in order to estimate area and performance.

Operating Systems

Our work in this area focuses on the development of Symunix, an operating system designed primarily for machines like the Ultracomputer (i.e., MIMD, shared memory, hundreds or thousands of processors, Fetch-and-Add, and hardware combining of memory references). The goal of this effort is to provide efficient execution of application programs. We use two techniques to accomplish this goal. First, we avoid, whenever possible, serial bottlenecks in the operating system and language runtime system. Second, we provide the necessary hooks in the operating system that permit a user-mode runtime system to perform tasks traditionally done by expensive traps into the kernel. Portability to architectures significantly different from ours is secondary.

Recent Accomplishments

During the past year we ported our stable version of Symunix from the 1985-vintage Ultra II prototype to the Ultra III prototype (currently under development with NSF funding). The porting process was valuable because it provided a stable component (the machine-independent bulk of the OS) to aid in ironing out problems with our software tools and the new hardware itself. We significantly upgraded our set of cross-development tools (compiler, linker, debugger), developed a deeper understanding of Ultra III machine dependencies, and helped to debug the hardware as new functionality was introduced.

We have nearly completed re-implementation of the process management and memory management portions of the kernel, consistent with the design presented in our 1991 proposal. The chief motivating factor is the need to provide more flexible and efficient support for programs with relatively fine grained parallelism. To that end, the new system provides asynchronous system calls and page faults, support for diverse memory sharing patterns, and provision for improved user/kernel cooperation in scheduling.

We have revised and expanded our collection of highly-parallel synchronization algorithms and data structures. They have been implemented with a very keen eye to efficiency (especially in the absence of contention) and memory usage. This is part of our campaign to attack the constants that limit the practicality of much highly-parallel computing research. One result of this effort is the evolution of families of algorithms for certain problems. Each algorithm has distinct semantic properties that set it apart from the others. For example, our newest kernel code incorporates algorithms for 9 different kinds of lists, differing in concurrency (serial, parallel), ordering discipline (FIFO, starvation free, starvation prone), memory consumption (constant, proportional to machine size), and support for special operations (interior removal, multi-insert). These distinctions are crucial for both correctness and performance.

One of the major questions we hope to answer through this research is, "at what size machine do Fetch-and-Add based algorithms such as readers/writers locks and highly-parallel queues begin to pay off?" Figure 1 shows some measurements that suggest the pay-off point may be lower than had been previously suspected. Four curves are presented to compare the performance of the Symunix *spawn* system call under three different kernel configurations. (Spawn is an extension of the traditional UNIX *fork*; it creates m new

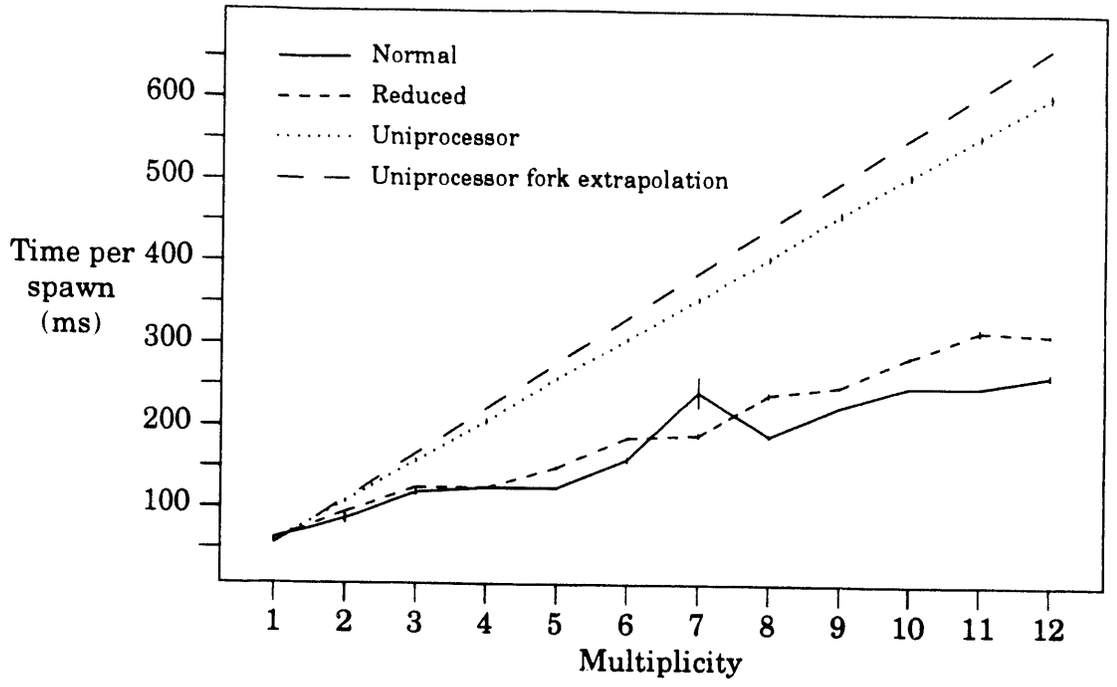

Figure 1: Cost of Spawn on Ultra II

processes. For the measurements shown in the figure, creation of a child includes copying 20K bytes of private data. The time for the parent to *wait* for the children's termination is also included.) In the "normal" case, highly-parallel synchronization algorithms and queues are used extensively. In the "reduced" case, the kernel was rebuilt to use only binary semaphores and lock-protected linked lists. In the "uniprocessor" case, further simplifications were performed to eliminate all multiprocessor synchronization; this version runs only on a single processor. In the "uniprocessor fork extrapolation" case, we simply show the expected cost of naively emulating spawn with fork (only the data for multiplicity=1 is real).

The graph shows a clear advantage for highly-parallel algorithms when the spawn multiplicity is 5 or more, except 7, where excessive swapping was caused by a weakness in the memory allocator (this will be addressed in the future). Of course, drawing conclusions at this point would be premature, as there are still some important questions: What is the effect of machine dependencies (e.g., Ultra II's small size, bus structure, and lack of combining)? How much does the "reduced" system's performance suffer, compared to an

implementation designed specifically for machines of Ultra II's scale? In practice, how significant are the constants affecting performance of highly-parallel algorithms?

Future Plans

In the next year we will build on our software development accomplishments, and concentrate on bringing Symunix to a level where significant application development and experimentation can be performed. Central to this will be integrating the new process and memory management structures into the rest of the kernel.

We will also test the Symunix approach to portability by addressing a second implementation target, the new series of NCR multiprocessors. Our approach to portability is based on a clear notion of the primary target architecture, combined with simple modularity. Conceptually, this works by starting with a design for a serial machine, and then adapting it to the Ultracomputer architecture; changes made are isolated by setting up simple modules with well-defined interfaces whenever possible. This strategy also facilitates experimentation, providing the opportunity to evaluate the significance of alternative module implementations. This was illustrated in the experiment of Figure 1.

The distinction between this approach and that of most other contemporary OS projects, is that we emphasize portability by modularity, instead of portable modules. For example, most other projects regard list structures as objects with portable implementations, or even as incidental *ad hoc* structures without any "object" or module status at all. In Symunix, we explicitly define many such module interfaces, with the intent that they be implemented differently on some machines. In addition, Symunix is an architecture-oriented system, clearly favoring a narrow class of machines. Nevertheless, many of the module implementations tailored to the Ultracomputer are still quite portable, in the sense that they will work correctly on a variety of other machines, even if performance suffers. Consequently, our effort to port to the NCR multiprocessor will proceed in stages. In the first stage, we re-implement only the minimum required modules. The second stage can be thought of as *tuning*, identifying and altering other modules to take advantage of the specific machine. It remains to be seen how effective this strategy will be at achieving good performance with a limited amount of module re-implementation. We plan to answer these questions next year.

One of the chief goals for Symunix is the flexible and efficient support of parallel program runtime support systems implemented primarily in user mode. This idea has also been recently expressed by Anderson *et al.* [1] and Marsh *et al.* [11]. The primary difference between these designs and ours (slightly extended from that presented in Edler *et al.* [6]) is our emphasis on avoiding serial bottlenecks. During 1992 we plan to prototype our approach by implementing a C-callable user-mode threads package, comparable in functionality to PCR [19], FastThreads [2], or CThreads [5]. Ultimately, we predict that the use of such packages will be limited sharply by the availability of programming languages with runtime environments tailored specifically to both language and architecture, but in the meantime a C-callable package can be an effective research tool.

As our hardware and software implementations begin to stabilize, we will be able to start collecting performance data. As already suggested, we will concentrate on evaluating the effectiveness of the highly-parallel software structures we employ, and also the impact of the combining network in the Ultra III prototype. These studies will be undertaken in concert with the simulation efforts to be begun next year.

The GNU C Compiler

Our efforts on the GNU C compiler over the past year have drawn the attention of a number of companies, several of whom have provided equipment for our use in this effort. Due to this interest, our work has expanded from providing the basic support for GCC that is required for our Ultra III prototype to producing commercial quality compilers for the IBM RS6000 and older PC RT as well as the 29000 (our original target). In addition, we are now the principal developers and maintainers of the back end of GCC.

Efforts over the past year include expanding optimizations to better meet the needs of modern RISC processors. These efforts, along with improvements to other optimizations, now result in GCC producing faster code for both the Sparc and Motorola 88000 than any other known C compiler, as measured using the SPEC benchmark.

We have worked with Cygnus Support, a commercial organization founded to support GNU software, in developing an instruction scheduler that tries to remove data dependencies between instructions that cause pipeline stalls. This scheduler separates memory

loads from uses of the register loaded, which increases the latency tolerance of the system. Tolerating memory latency is particularly important for shared-memory multiprocessors.

Efforts in subsequent years will be similar: we will continue improving the optimization abilities of GCC in addition to supporting the compiler for both local and external users. For use with our prototypes, we plan to include support for a *shared* keyword and to allow selected optimizations on *volatile* variables. It is likely that coordination variables will be declared volatile. We are also enhancing the backend of GCC to support the GNU FORTRAN effort. The result will be a freely available, high quality FORTRAN compiler that we will provide to users of our prototypes.

The high quality of GCC has attracted the attention of the Ada language community. The NYU Ada group is seeking support for their effort to produce a GNU Ada compiler.

Simulation

Recent Accomplishments

Since the project's inception, we have continually built and used simulators to study multiprocessor performance. Simulations driven by synthetic reference streams have been used extensively by our group and by others for studying the performance of multiprocessor networks. In contrast to these simulators, we also built an address trace driven simulator to study the performance of multiprocessor TLBs.

Results of our trace-driven simulations allowed us to compare the performance of multiprocessor TLBs located (unconventionally) at memory with that of TLBs located at processors. As described in Teller and Gottlieb [18], for the systems and workloads studied, TLBs located at memory perform better than TLBs located at processors, provided that memory is organized as multiple clusters of memory modules, where the mapping of a page is fixed to a particular cluster and the page is interleaved across the cluster's memory modules. The cost of a processor-based TLB reload is at least $\log N$ because of network transit time. The cost of a memory-based TLB reload can be made smaller than that of a processor-based TLB reload since network transits are not required. Furthermore, with multiple clusters of memory modules, the number of reloads is smaller with memory-based TLBs than with processor-based TLBs. For memory-based TLBs to continue to outperform processor-based

TLBs for large N, it is likely that the number of clusters must grow with N.

Due to limitations in our simulation environment, we were able to collect address traces for only applications with a fixed and restricted problem size. The problem size restriction limited the maximum available parallelism and, thus, the number of processors that could cooperate in program execution, i.e., the effective size of a multiprocessor system. In addition, it did not allow us to increase the size of the problem with the size of the scalable multiprocessor, as we would have liked to.

Without being able to simulate larger systems, we were not able to determine if the number of TLB reloads for the workloads studied grows linearly with N. We plan to study these and other workloads in our new simulation environment, described below, that will allow us to run larger problem sizes with higher levels of parallelism.

The programs used in our study proved to be poor candidates for demand paging. Even when the number of frames allocated permitted us to store half of the shared read-write pages, many page faults occurred and performance was significantly degraded. We are pursuing how prevalent this phenomenon is in other "dusty deck" scientific programs.

We intend to continue our study of TLB reload and other overheads since a large reload overhead that grows strongly with N will have a deleterious effect on speedups attained for programs executed on highly-parallel multiprocessors.

Future Plans

To continue this research and to perform new research that depends on simulations, we are currently in the process of building a flexible multiprocessor simulation environment, similar in some ways to Tango [3], that will allow us to perform either execution- or trace-driven simulations. This environment will allow us to simulate a multiprocessor in detail or to simulate specific subsystems in detail, while others merely are modeled. The instruction-level simulator for the AMD 29000, the multilevel network simulator mentioned in the VLSI section, and the trace-driven simulator mentioned above all will be included in this environment.

Such a simulation environment will allow us to study the performance of components and subsystems of multiprocessors, in addition to continuing our study of network and TLB performance. In particular, we would like to study the performance of different size

Ultracomputers and the effect of combining, to characterize parallel programs, to compare different parallel algorithms and different methods of ensuring cache consistency, and investigate the feasibility of demand paging.

Coordination Algorithms

Recent Accomplishments

One line of recent study has been the usefulness of Fetch-and-Increment and Fetch-and-Decrement. To our surprise, we discovered that these two primitives can be utilized to implement many coordination algorithms (most notably reader-writer locks) with asymptotic complexity comparable to that achievable with the more general Fetch-and-Add. These results were presented at ASPLOS-IV (Freudenthal and Gottlieb [7]).

Although equal in asymptotic complexity, Fetch-and-Increment algorithms require more accesses to shared memory than their Fetch-and-Add counterparts for the majority of problems we have examined. This observation is part of our overall program to "attack the constants", i.e. to improve performance by lowering the constants that are often swept under the asymptotic rug. The choice between a machine with Fetch-and-Add, the architecture we still favor, or the simpler Fetch-and-Increment is more complex than we first believed.

Recently, we have experimented with efficient mutual exclusion algorithms that guarantee requests are satisfied in first-come first-served (i.e. fair) order. One such algorithm requires only a single access to shared memory in order to request or release a semaphore in the absence of contention. Polling, which would otherwise occur in the presence of contention, can be avoided by utilizing the Ultracomputer *reflect* operation, which allows a processor to send a trigger to any other processor.

Future Plans

Extending the work accomplished this past year, we will explore algorithms for efficient, fair mutual exclusion for processes with priorities. Such algorithms require the development of an efficient priority queue mechanism, which is also being investigated.

BBN permitted us to use one of their TC2000s to evaluate the comparative performance of our many parallel queue algorithms in an environment with significantly more than eight processors. These tests were inconclusive since Fetch-and-Add on the TC2000 requires an expensive system call, making the results hard to interpret. We will perform these experiments again in 1992 on the 16-processor Ultra III prototype and on the Ultracomputer simulator described in the simulation section.

We will also evaluate the performance of several of our new coordination algorithms via simulation and execution on prototype hardware. In particular, our broadcast trees (for asynchronous signal delivery) and prefix multiqueues are nearly ready for execution and evaluation.

We have been following the recent work of Mellor-Crummey and Scott [13] emphasizing that contention due to spin-waiting on synchronization variables in remote memory can cause substantial performance degradation on several of the shared-memory computers currently available. For that reason, they advocate constructing machines with shared, processor-local memory and using algorithms that poll shared local variables.

We believe that this degradation will be minimized by the combining and buffering features of the Ultracomputer's network that are not present on the machines they examined. In addition, as mentioned above, the reflect operation can be used to replace polling of a shared variable. We intend to quantify, using simulation, the algorithmic tradeoffs involved in hardware vs. software combining and in local polling vs. remote polling vs. reflect.

Mellor-Crummey and Scott studied semaphores and barriers, the latter requiring trees similar to those used in software combining. We plan to study how widely applicable these techniques are and what additional data structures they require when applied to other problems. It is as easy to support shared local memory in the Ultracomputer as in any other shared-memory architecture; for all such architectures, the cost of dual porting the memories must be considered.

Applications

The applications work reported is supported jointly by our program and by the DOE Mathematics program at NYU, supervised by Peter Lax.

Recent Accomplishments

During the past year, we worked with L. Greengard of CIMS and G. B. McFadden of NIST to develop a fast solver for Laplace's equation on multiply connected domains. The method uses a well-conditioned second kind Fredholm integral equation to represent the solution. This equation is discretized using the Nystrom method with the trapezoid rule, resulting in a dense, nonsymmetric system of linear equations to solve. Fortunately, most of the eigenvalues of this linear system are tightly clustered around one, making it amenable to fast solution by iterative methods. We used the GMRES method (generalized minimum residual method), which is most suitable for problems of this sort. A preconditioner was also developed that enabled fast convergence even when the GMRES method is restarted. The bulk of the work at each iteration is in applying the dense matrix to a vector. This was accomplished using the adaptive fast multipole method, a procedure for computing the product of an n by n Cauchy matrix with a given vector in time $O(n)$. The combination of these numerical methods brings computations on extremely complex domains within practical reach. Figure 2 shows a region with 200 holes, on which we solved the infinite exterior Dirichlet problem:

$$\Delta u = 0, \quad \textit{outside the holes}$$

$$u = \textit{curvature}, \quad \textit{on the boundaries.}$$

This computation required 13 GMRES iterations to achieve 6 decimal digit accuracy in the linear system solve and was completed in under an hour on a Sparcstation 1. Had the same algorithm been used without the fast multipole method (using a standard dense matrix vector multiply procedure), the time required would have been about 17 hours. If one attempted to solve the dense 20200 by 20200 linear system (generated using 100 discretization points per boundary) using Gaussian elimination (assuming this matrix actually could be stored in memory), the time required would have been about 3.6 months! Thus, the combination of all of these techniques is required if computations (especially time dependent ones, as will be described later) are to be feasible on such domains.

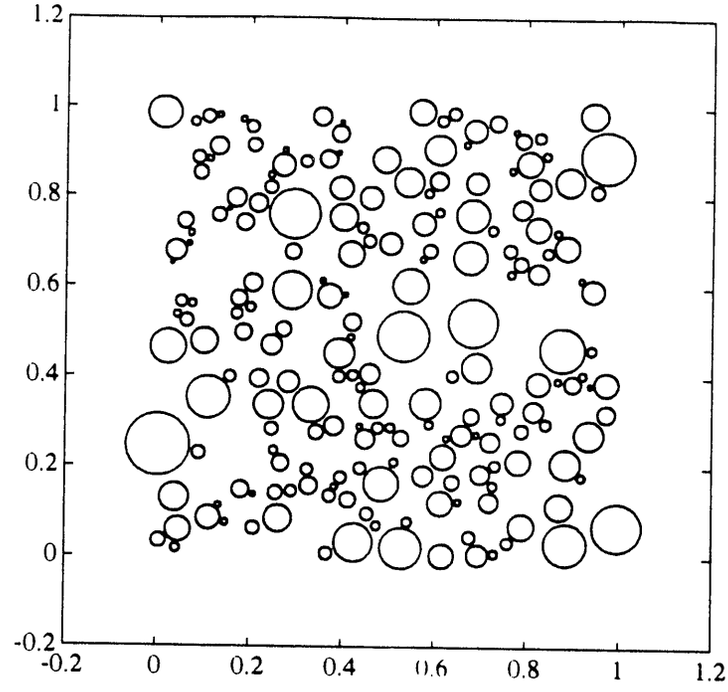

Figure 2: Region with 200 Holes

The computational methods developed here and similar ideas are applicable to a wider range of problems than simply Laplace's equation. We have also worked with A. Mayo, of IBM T. J. Watson Research Center, and with L. Greengard in applying similar techniques to solve the biharmonic equation. This is a more difficult equation to solve, and early work indicates that iterative methods such as GMRES require significantly more iterations to converge. Work on this problem is continuing.

Other work over the past year has been of a more general nature, trying to understand the reasons iterative methods behave as they do on these types of problems and developing techniques to improve that behavior.

Future Plans

The development of an efficient method for solving Laplace's equation on multiply connected domains, described previously, is a necessary first step in solving time dependent problems modeling particle coarsening during Ostwald ripening (McFadden [12]). In the

next year, we plan to incorporate our fast Laplace solver into a 2-D time dependent simulation of this process. This phenomenon occurs when bodies of one material are embedded in another material of different thermodynamic phase. Material diffuses from one body through the surrounding medium to another body in such a way as to minimize the area of the interface. The motion of a point z on the interface boundary is described by an ordinary differential equation in time:

$$\dot{z} = u_n \cdot \vec{n}(z,t)$$

where $\vec{n}$ is the unit outward normal at the point z, and u satisfies

$$\Delta u = 0, \quad \textit{outside the bodies}$$

$$u = \textit{curvature}, \quad \textit{on the boundaries.}$$

The techniques developed over the past year will be used to evaluate the right-hand side of this ordinary differential equation.

It is not clear what the best method is for solving this system of ordinary differential equations. Explicit methods are limited to time steps of order h^3, where h is the minimum distance between spatial discretization points (McFadden [12]). Implicit methods require solution of a linear system with the Jacobian matrix at each iteration of a Newton-like procedure. The Jacobian matrix is dense and should be solved with an iterative method, but a good preconditioner will be required. It is unknown how to derive such a preconditioner. Other issues include regridding and removal of bodies when they become too small. It is an open analytic question to determine the effect of the removal of a small body on the system as a whole. It is also not clear if tracking boundary points is the best way to do the numerical simulation. Other ideas have been proposed (e.g., Osher [14]), and these methods are being considered as well.

The application of similar techniques in other areas is also being planned for the coming year. As mentioned previously, work on the biharmonic equation has begun. We also plan to apply these ideas to problems in conformal mapping. Capacitance calculations are another area of interest requiring similar techniques. Finally, we are studying the properties required of a nonsymmetric linear system solver to perform efficiently on this class of problems. We believe a variant of the restarted GMRES method can be made to perform efficiently on problems with just a few outlying eigenvalues, which is characteristic of the matrices arising from second kind integral equations.

Applied Probabilistic Analysis

Our work in applied probabilistic analysis involves two areas:

- Constructing and solving carefully chosen models to illuminate the behavior of the clock-regulated queues and switches in the Ultracomputer interconnection network.
- Designing and testing pseudo-random number generators for current and future parallel processors.

Recent Accomplishments

We have refined our analysis of the single-stage behavior of two practical models for a 2×2 switch and verified it with simulations [4]. Each of these models represents a switch design using queues that can accept only one input at a time. One, which models the switch chip we have fabricated, uses four queues, one for each input/output pair; the other models a design using only two queues, one at each input. The analysis substantiates the intuition that the four-queue design has much better performance under moderate and heavy loads.

A number of authors have investigated parallel computations that use the same linear congruential generator on each processor but with widely spaced seeds, apparently without realizing that, in achieving good short range independence, such linear congruential generators are designed with long range dependencies (Marsaglia [10], Percus and Percus [15]). We have recently shown [16] two different general relationships that exist among terms of the linear congruential generator

$$x_{(i+1)} = ax_i + b \ mod \ 2^\beta$$

that are separated by powers of two. These two relationships are not equivalent except in one special case which turns out to be Marsaglia's result.

Future Plans

In the area of interconnection network modeling we are currently considering the following problems:

- Extending the results in Percus and Percus [17] to a characterization of queueing behavior in network stages beyond the second stage.

- Using difference equations we have developed to compute the time-dependent queue length and output rate probabilities for switches with 2-way and unlimited combining, and verifying this with simulations.
- Analyzing the queueing behavior and time series transformations in 4×4 and higher degree switches with clock-regulated queues.
- Exploring the conditioning of the traffic by the forward interconnection network from processors to memories that, according to simulations by Liu [9], appears to result in less queueing delay on the return network from memories to processors.

Our study of pseudo-random number generators represents a continuing investigation of a fairly broad topic. There are a number of threads that can and will be considered in parallel with the expectation that each thread will cast light upon the others. In general, the research will fall into the following three categories:

- The development of tests specifically geared to the problems inherent in parallel computations. When many realizations of a simulation are run in parallel, it is imperative that the sequence of pseudo-random numbers on each processor be independent. Ideally, there should be no correlation between the sequences on different processors. An amalgamated sequence made by combining the sequences of all n processors must be tested by suitably devised new single sequence tests.
- Continued investigation of linear congruential generators for parallel processors with different values of the multiplicative constant a. Previously we have investigated linear congruential generators with different additive constants b [8]. Variants of parallel generators with different a's instead of or in addition to different b's may have superior statistical properties.
- Extension of theoretical investigations to other types of generators for use in parallel environments. We will attempt to generalize the analysis of sequential generators beyond that of the linear congruential generators, e.g. to the generalized feedback shift register and lagged Fibonacci generators.

References

[1] Thomas E. Anderson, Brian N. Bershad, Edward D. Lazowska, and Henry M. Levy, "Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism," *Proc. 13th ACM Symp. on Operating Systems Principles (SOSP)* (October, 1991).

[2] Thomas E. Anderson, Edward D. Lazowska, and Henry M. Levy, "The Performance Implications of Thread Management Alternatives for Shared Memory Multiprocessors," *IEEE Trans. on Comp.* **38** (12) pp. 1631-1644 (December, 1989).

[3] Helen Davis, Stephen R. Goldschmidt, and John Hennessy, "Multiprocessor Simulation and Tracing using Tango," *Proc. 1991 Inter. Conf. on Parallel Processing*, pp. H-99-H-107 (August, 1991).

[4] Susan R. Dickey and Ora E. Percus, "Performance Analysis of Clock-regulated Queues with Output Multiplexing in Three Different 2 by 2 Crossbar Switch Architectures," *J. of Parallel and Distributed Comp.* (to appear).

[5] R. Draves and E. Cooper, "C Threads," Technical Report CMU-CS-88-154, School of Computer Science, Carnegie-Mellon University (June, 1988).

[6] Jan Edler, Jim Lipkis, and Edith Schonberg, "Process Management for Highly Parallel UNIX Systems," *Proc. USENIX Workshop on UNIX and Supercomputers* (September, 1988).

[7] Eric Freudenthal and Allan Gottlieb, "Process Coordination with Fetch-and-Increment," *Proc. 4th Inter. Conf. on Architectural Support for Programming Languages and Systems (ASPLOS-IV)*, pp. 260-268 (April, 1991).

[8] Malvin H. Kalos and Ora E. Percus, "Random Number Generators for MIMD Parallel Processors," *J. of Parallel and Distributed Comp.* **6** pp. 477-497 (1989).

[9] Yue-sheng Liu, *Architecture and Performance of Processor-Memory Interconnection Networks for MIMD Shared Memory Parallel Processing Systems,* Ph. D. Dissertation, New York University (May 1991).

[10] G. Marsaglia, "The Structure of Linear Congruential Sequences," pp. 249 in *Application of Number Theory to Numerical Analysis*, ed. S. K. Zaremba, Academic Press (1972).

[11] Brian D. Marsh, Michael L. Scott, Thomas J. LeBlanc, and Evangelos P. Markatos, "First-Class User-Level Threads," *Proc. 13th ACM Symp. on Operating Systems Principles (SOSP)* (October, 1991).

[12] G. McFadden, P. Voorhees, R. Boisvert, and D. Meiron, "A Boundary Integral Method for the Simulation of Two Dimensional Particle Coarsening," *J. Sci. Comp.* **1** (2) pp. 117-144 (1986).

[13] John M. Mellor-Crummey and Michael Scott, "Algorithms for Scalable Synchronization on Shared Memory Multiprocessors," *ACM Trans. on Comp. Systems* **9** (1) pp. 21-65 (February, 1991).

[14] S. Osher and J. Sethian, "Fronts Propagating with Curvature Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations," *J. Comp. Phys.* **79** (1) pp. 12-49 (1988).

[15] Ora E. Percus and J. K. Percus, "Long Range Correlations in Linear Congruential Generators," *J. Comp. Phys.* **77** pp. 267-270 (1988).

[16] Ora E. Percus and J. K. Percus, "Intrinsic Relations in the Structure of Linear Congruential Generators modulo 2^{β}," Ultracomputer Note #172 (January, 1991).

[17] Ora E. Percus and J. K. Percus, "Time Series Transformations in Clocked Queueing Networks," *Comm. Pure Appl. Math.* (1991).

[18] Patricia J. Teller and Allan Gottlieb, "TLB Performance in Multiprocessors," NYU Ultracomputer Note #173, New York University (September, 1991). Submitted for publication in special issue of the *J. of Parallel and Distributed Comp.* on Memory System Architectures for Scalable Multiprocessors.

[19] Mark Weiser, Alan Demers, and Carl Hauser, "The Portable Common Runtime Approach to Interoperability," *Proc. 12th ACM Symp. on Operating Systems Principles (SOSP)*, pp. 114-122 (December, 1989).

END

DATE
FILMED
12/27/91