1. Project Summary

The NYU Ultracomputer project has studied the scalability of computing systems and algorithms, specifically the shared-memory computation model. Currently under construction is a 16-processor Ultracomputer using NYU designed, custom VLSI switches to interconnect processors and memories. This system supports fetch-and-phi coordination primitives and is the first to support asynchronous combining of simultaneous requests to the same memory location.

The present proposal embarks on a program of characterizing and modeling the performance and behavior of scalable shared-memory systems and their algorithms. Numerous measurements and architectural experiments will exploit the configurability of the 16-processor prototype and a 256-processor system that may become available during the proposal period. These systems have a number of novel features.

- The PEs support a large variety of fetch-and-ϕ operations, which are used in synchronization algorithms.
- Each PE contains a low-latency network interface. In the absence of contention, all operations complete in 13 clock cycles (under 1μs), including network transit time, which grows by two cycles for each doubling of the system size.
- The interconnection network is a high-bandwidth network containing NYU-designed custom CMOS VLSI chips that buffer requests and responses between PEs and MMs. If two requests for the same memory location are present in the same network node, they are combined into one request, thus removing serial bottlenecks when synchronization variables are accessed.
- The prototype is designed to enable performance monitoring and to simplify performing architectural experiments. Most of the logic on the PE is contained in two XILINX field-programmable gate arrays (FPGAs). These chips contain a configuration RAM used to specify the function performed by each computation block on the chip and the interconnections of these blocks. This RAM is down-loaded from a PC. The network boards contain wiring reserved for the addition of a XILINX chip to be used only for measurements.

The use of XILINX chips in the PE provides two capabilities that will enhance experimentation. First, the PEs have the ability to count a large number of arbitrary events within the PE and to produce summaries such as histograms of network delays. Second, various parameters of the network interface can be adjusted to determine the effects of these choices on system performance, on both a micro- and macroscopic level. The presence of XILINX chips on the switch board enables monitoring of the behavior of the network and recording statistics such as the number of combined operations at each switch and maximum and typical queue lengths within the switches.

These tool enable the following evaluation strategy:

- Evaluate the impact of combining on overall system performance.
- Characterize the behavior of applications on the Ultracomputer architecture. Scientific applications will be studied in detail.
- Examine issues in the design of parallel operating systems, especially synchronization algorithms.
- Explore alternatives in processing element design.
- Study the behavior of the multi-stage interconnection network.
- Construct mathematical models of parallel system behavior based on our experiments and measurements. Such models can be used to predict the behavior of large numbers of parallel processors.

2. Project Description

2.1. Introduction

For the last decade, the NYU Ultracomputer project has been researching the scalability of computing systems and algorithms, specifically the shared-memory computation model. We have investigated many aspects of this model, from custom VLSI design of components for a processor-memory interconnection network to studies of variable-grid methods in scientific computation. Our current NSF grant is funding the construction of a 16-processor Ultracomputer using our custom VLSI switches to interconnect processors and memories. This system supports fetch-and-phi coordination primitives and is the first to support asynchronous combining of simultaneous requests to the same memory location [10].

The present proposal embarks on a program of characterizing and modeling the performance and behavior of scalable shared-memory systems and their algorithms. Numerous measurements and architectural experiments will exploit the configurability of our 16-processor system and a 256-processor system that may become available during the proposal period.

The prototype we are currently completing consists of 16 processing elements (PEs) based on the AMD Am29050 processor, 16 memory modules (MMs), each containing 16MB of DRAM, and a 16×16 interconnection network. This prototype has a number of novel features:

- The PEs support a large variety of fetch-and-ϕ operations, which are used in synchronization algorithms.
- Each PE contains a low-latency network interface. In the absence of contention, all operations complete in 13 clock cycles (under 1μs), including network transit time, which grows by two cycles for each doubling of the system size.
- The interconnection network is a high-bandwidth network containing NYU-designed custom CMOS VLSI chips that buffer requests and responses between PEs and MMs. If two requests for the same memory location are present in the same network node, they are combined into one request, thus removing serial bottlenecks when synchronization variables are accessed.
- Our prototype is designed to enable performance monitoring and to simplify performing architectural experiments. Most of the logic on the PE is contained in two XILINX field-programmable gate arrays (FPGAs). These chips contain a configuration RAM used to specify the function performed by each computation block on the chip and the interconnections of these blocks. This RAM is down-loaded from a PC. The network boards contain wiring reserved for the addition of a XILINX chip to be used only for measurements.

We have recently constructed a 4-PE system using all the major components required for the 16-PE prototype. All that remains is to have more PE and MM boards constructed and to have NCR, one of our industrial partners, fabricate the custom VLSI chips for our interconnection network using a more advanced process and higher pin-count packages than those available to us through MOSIS (the current, fully-functional 4×4 network uses MOSIS-fabricated chips). We expect the full system to be completed during the first three months of 1993. A proposal has been submitted to DARPA for MCC, another industrial partner, to construct a 256-PE Ultra III. If funded, this prototype will available for study near the end of 1994.

The use of XILINX chips in our PE provides two capabilities that will be used by our experiments. First, we have the ability to count a large number of arbitrary events within the PE and to produce summaries such as histograms of network delays. Second, we can adjust various parameters of the network interface to determine the effects of these choices on system performance, on both a micro- and macroscopic level. The presence of XILINX chips on the switch board enables us to monitor the behavior of the network and record statistics such as the number of combined operations at each switch and maximum and typical queue lengths within the switches. A detailed discussion of these features is given in Section 3.

Using these tools, we will pursue the following evaluation strategy:

- Evaluate the impact of combining on overall system performance.

- Characterize the behavior of applications on the Ultracomputer architecture. Scientific applications will be studied in detail.
- Examine issues in the design of parallel operating systems, especially synchronization algorithms.
- Explore alternatives in processing element design.
- Study the behavior of the multi-stage interconnection network.
- Construct mathematical models of parallel system behavior based on our experiments and measurements. Such models can be used to predict the behavior of large numbers of parallel processors.

An outline of the experiments to be performed in each of these areas is given in Section 4. These experiments may uncover unanticipated performance problems or possible architectural enhancements. Later experiments will be designed to answer questions raised by previous work.

Three main groups of researchers will be performing this ambitious measurement and characterization work:

(1) A core group, consisting mostly of the full-time staff that produced the hardware and software of our prototype, will be responsible for developing and maintaining instrumentation hardware, firmware, and software. They will also be responsible for maintaining the hardware and software of the prototype and performing a ‘‘user services’’ function.

(2) A measurement group, consisting mostly of graduate students and their advisors, will perform most actual measurements and data collection.

(3) A cross-disciplinary group, consisting of numerical analysts and researchers from the natural sciences, will provide the application programs to be used in our study. Mathematicians will also assist in our modeling program.

We expect our 16-PE machine to have an aggregate performance high enough to attract real users. The scientific applications used in our study will come from this group of users. In order to provide adequate availability of our prototype, we will construct a 4-PE system during the first year for firmware and software development. Our equipment budget includes the funds for this construction in addition to funds for any new instrumentation we require.

The remainder of this proposal contains detailed information on our 16-PE prototype (Section 2), the measurement and configurability capabilities of our hardware (Section 3), a discussion of our evaluation strategy and goals (Section 4), and our accomplishments to date (Section 5).

2.2. Ultra III Design

The Ultra III system is composed of 16 Processing Elements (PEs), packaged four to a board, 16 Memory Modules (MMs), similarly packaged, and a 16×16 combining network, packaged in 8 boards, each consisting of a 4×4 fragment. Each component is described below.

2.2.1. Processing Element

The primary goal of our PE design was support for all the components of the Ultracomputer architecture. These capabilities had to be supported in a way that did not make their use *preferentially inefficient*; for example, supporting fetch-and-add only via a system call would discourage its use. Specifically, the architectural features supported are:

- fetch-and-ϕ operations available to application-level code.
- multiple simultaneous operations outstanding in the network.
- ‘‘fence’’ operations used to enforce sequential, and weaker, consistency models.
- application control of cachability and explicit cache invalidations to support software-controlled cache coherence.

Supporting these features without a performance penalty requires an efficient network interface. To achieve this goal, a request starts being transmitted to the network one to three cycles after it is made by the processor. A PE with high performance encourages application developers to run their applications on our prototype, thus providing more ‘‘test data’’ for experimentation.

The remaining goals for the PE design were simplicity, measurability, and configurability. The design of a PE using the Am29050 is discussed in this section and is shown in Figure 1.

The Am29050 was chosen because of two features that we exploit in our PE design. First, four user-settable bits are present on every **load** and **store** instruction. Most encodings of these bits are not defined by the processor and are reserved to external hardware. We use these bits to support our fetch-and-ϕ operations.

Second, the Am29050 has features that minimize the effects of long memory latency for both instruction and data fetches. It prefetches instructions into a four-word buffer, essentially pipelining instruction fetches. A branch target cache is also provided on the chip. The bus provides mechanisms that allow simultaneous instruction and data references. Once a data load is issued, execution can proceed until either the register is read or another load is issued. In the latter case, the second load is placed on the bus as a pipelined request, which we can issue to the network while the first load is still pending.

The PE consists of the Am29050 processor, instruction and data caches, a network interface, and control logic. The control logic consists of two major parts, one for instruction and one for data. Each ‘‘side’’, instruction and data, has its own state machine and cache. These state machines are duplicated in both XILINX chips and are responsible for determining hit or miss, making network requests, and handling responses from the network. Arbitration is performed at the outgoing network port, which also assigns an ORI value to each outgoing request; responses are directed to the appropriate side by consulting the ORI. The data side permits up to eight outstanding stores and supports two pipelined loads; the instruction side prefetches an entire cache line from the network.

The caches are direct-mapped and write-through. We use the IDT7174 chip, which combines static RAM with a comparator for the tag storage, and conventional SRAMs for data storage. Each cache contains 8K lines, the size of the tag store chips. The instruction and data caches have different goals. On the instruction side, it is important to provide 32 bits of instructions each cycle whenever the accesses are sequential (made using the Am29050's burst mode). Therefore a line size of 128 bits is used.

For data caches, burst mode accesses are relatively rare and 32-bits per cycle bandwidth is not required. Also, straightforward implementation of software consistency strategies requires each cache line to consist of only one word. This is also the simplest design since it does not have to deal with partially-valid lines (which cannot occur with the read-only instruction cache). Therefore, the data cache is 8K$\times$32 bits.

The control logic, including both state machines, and some of the data paths is implemented using XILINX programmable gate arrays; two 4000-series chips are used on each PE. Using programmable logic enables a large variety of experiments and measurements to be performed, as will be discussed in Section 3.

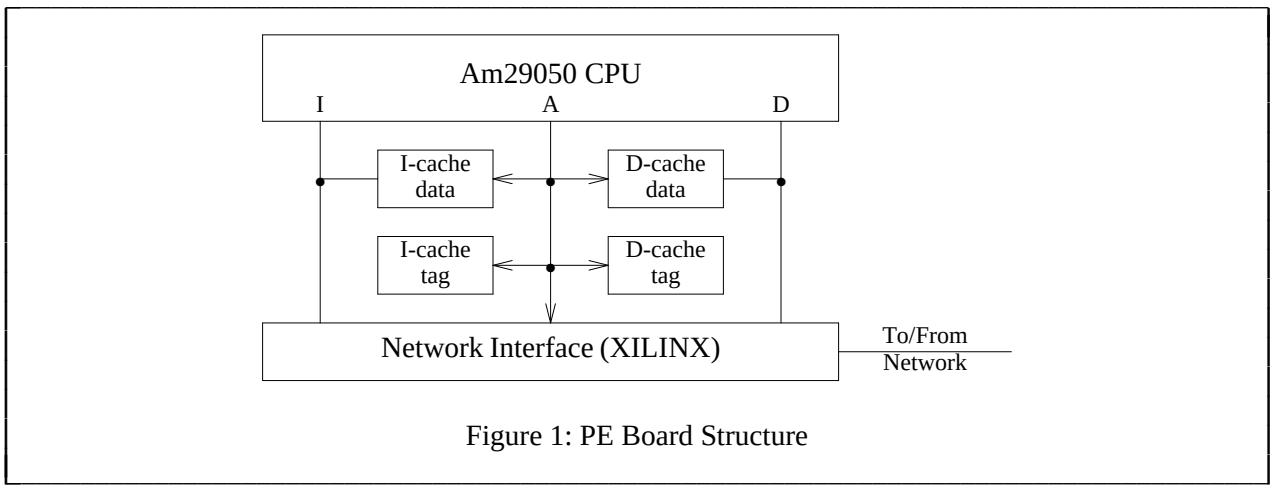

Figure 1: PE Board Structure

2.2.2. MM Design

The major function of the MM is straightforward: it receives requests from the network and responds to them. (The MM also has a second port that connects to a back-end I/O processor. Because of the pipelined nature of the entire system, arbitration between the ports is not in the critical path and can be done in parallel with servicing requests.)

Since the PE requests include fetch-and-add, the memory also contains a 32-bit ALU. Overall performance is influenced by the MM's cycle time (the reciprocal of its service rate). Specifically, the MMs must be able to provide an aggregate service rate greater than the aggregate request rate of the PEs (or more specifically the caches). In contrast, any latency added by the MM will be small compared to network latency. That is, it will be the memory cycle time and not its access time that will most strongly determine performance.

Memory cycle time is the reciprocal of the clock rate times the number of cycles needed to process a request. The Ultra III prototype has a clock rate of 15 MHz, i.e., packets arrive every 67 ns. Since the network requires all requests to contain an even number of packets, messages can arrive every 133 ns. The major limitation on the MM cycle time is that of the memory chips. The use of relatively inexpensive DRAMs provides MM cycle times in the range of 150-200 ns while allowing the MM to process a request in four network clock cycles. Simulations have shown this cycle time to be adequate. The instruction cache issues load double requests, which generate four packet responses. Since the MM is organized as a 4Mb×32 (plus parity) array, a 64-bit load requires two accesses. Using page mode to provide the second 32-bit word permits the DRAMs to complete these loads in four cycles as well. The MM begins transmitting the first packet of the response two cycles after the adjacent switch begins transmitting the first packet of the request to the MM.

To permit loads to be combined with fetch-and-adds, the PEs transmit 32-bit loads as fetch-and-adds with an increment of zero; this does not increase network traffic since our switches require all messages to have an even number of packets. Similarly, a 32-bit store is transmitted as a fetch-and-store. The full list of operations supported by the memory are: Load Double, Store Double, Fetch-and-Add, Fetch-and-Or, Fetch-and-AndNot, Fetch-and-Store, Fetch-and-Store-if-Zero, Fetch-and-Store-if-≥0, Partial-Word-Store, Reflect, and Broadcast. All operations are combinable by the network except for Partial-Word-Store, Reflect, and Broadcast.

2.2.3. Interconnection Network

The Ultracomputer network is built from $K \times K$ switches and has the topology of the omega network [16] and the SW Banyan [9]. It consists of $\log_K N$ stages, where N is the number of PEs in the system. The Ultra III network has $\log_2 N$ stages, each having $N/2$ 2×2 switch nodes that routes memory requests from PEs to MMs and responses (to these requests) from MMs to PEs. A switch has eight unidirectional ports, four of which are used to transmit requests from PEs to MMs and four that transmit responses back to PEs.

Each 2×2 combining switch node is composed of two parts: a *forward path component* (FPC) and a *return path component* (RPC). Using the 208-pin packages furnished by our industrial partner NCR, each component requires two (identical) chips. The FPC contains four combining queues, one for each input-output pair, that combine requests having the same operation and memory address. The merged request is sent towards the memory and the older of the two original requests is sent to the RPC where it is placed into a *wait buffer*. Each memory response is compared against all entries in the appropriate wait buffer and then queued for output back toward the originating PE. Any matched message from the wait buffer is similarly queued. The RPC contains two wait buffers, one for each MM port, and eight (noncombining) queues, two for each input-out pair (one for memory responses and one for matched entries from a wait buffer).

Each message includes routing information; routing decisions are made independently by each switch. Flow control avoids the necessity of acknowledgement for messages and allows pre-computation of signals that control data movement from stage to stage. The components accept both two and four packet messages. Packets that arrive when the queue is empty and not blocked exit at the next cycle. Included on each 4×4 board is space and wiring to enable a XILINX 4000-series chip to observe all the

control signals for measurement purposes.

2.3. Instrumentation Tools

The Ultra III prototype is designed to permit extensive measurements and the adjustment of system parameters. This section discusses those facilities and how we will use them.

The list of experiments in this section is not meant to be all-inclusive. A major part of our research is characterizing the system: we expect to discover many new things to measure and modify. Early experiments will likely motivate subsequent ones. The XILINX chips being used are highly configurable and thus are unlikely to be a major limitation on which measurements we can perform.

In the discussion below, we try to indicate the relative difficulty of the instrumentation techniques. In some cases, they are fairly easy and will definitely be performed, while in other cases, we may discover that our time is better spent doing a larger number of simpler experiments that only indirectly obtain the same information.

The PE has significant ability to instrument its performance and adjust system parameters, the switch has extensive monitoring ability and a smaller ability to modify its performance, and the memory has few capabilities in both areas. Each of these will be discussed in turn, preceded by a discussion of general measurement issues.

2.3.1. External Measurements

Even without the extensive monitoring ability and configurability built into our hardware, there are some basic evaluation tools available and we will start by making extensive use of them. The most fundamental tool is, of course, the ‘‘stopwatch’’ and the most fundamental configuration parameter is the number of PEs running an application. Most investigations will start with obtaining gross performance and speedup measurements.

Experience with our earlier Ultra II prototypes has shown the importance of having a front panel containing lights that monitor various aspects of the system such as processor stall, memory activity, and activity at each switch. Such a panel for the Ultra III prototype is under construction as of this writing. Panels of these kind, while somewhat crude and by nature imprecise, have proven to be extremely useful in visualizing what the machine is doing at any point in time. Once a gross characterization of the performance of an application or algorithm has been obtained, observing the lights while the program is running often shows periods of time during which the program serializes and gives initial indications of how well the network is performing during execution. Any anomalies observed at this time will be subjected to the detailed analysis made possible by the features described in the remainder of this section.

2.3.2. PE Instrumentation

Most of the logic on the PE is contained in two XILINX 4000-series programmable gate arrays (the exception being some critical-path logic involving control of the Am29050 bus that is done by fast PALs). The XILINX chips are downloaded from a PC during system boot, and can also be dumped to the PC. They contain thousands of gates, divided into hundreds of Computational Logic Blocks (CLBs) and routing resources to connect the CLBs. Virtually all of the control signals on the PE are observed or generated by these chips. They also provide the network interface.

This concentration of all the key signals and control logic in two field programmable logic arrays allows extensive measurement capabilities along with the ability to modify the PE functionality to a great extent. These are discussed below, in turn.

2.3.2.1. PE Measurements

The most basic measurement one can make is to count the occurrence of various events over a period of time. Most of our measurements on the PE will be of that form. Ripple counters are very cheap in XILINX chips, taking only 1/2 CLB per bit. For a small number of counters, sufficient routing resources are available within the XILINX to read these out onto the processor data bus (which is wired to one of the XILINX chips), though counting must be suppressed for a few cycles to allow the counter to settle.

With a larger number of counters, the data can only be obtained by shutting down the machine and reading the state of the XILINX via its diagnostic port.

In addition to the information obtained from bus control signals, the Am29050 has external status lines that give the state of the processor and can be decoded to produce additional events. The following is a partial list of the kinds of events we can count by straightforward observation of both the bus control signals and the processor status lines:

- instruction fetches, nonsequential i-fetches (branches), i-cache hits.
- data requests, pipelined data requests, d-cache hits, writes.
- occurrences of each fetch-and-ϕ operation.
- broadcasts and reflects received.
- cycles that the processor is in pipeline hold.
- cycles in supervisor mode, cycles processing traps.
- times that more than N stores are outstanding to the network.
- times that simultaneous i-fetches and d-fetches are outstanding.

By writing more complex XILINX ‘‘microcode’’, we can record the number of cycles between a request and response, record minimum, maximum, and average delays, and prepare a histogram of delays over some (small) range. Variations of this technique also allow us to measure other types of latencies.

2.3.2.2. PE Configurability

Many experiments will be performed by varying the behavior of the processor and seeing the impact on execution of algorithms and applications, for example:

- Disabling instruction and/or data caches.
- Increasing or decreasing the amount of prefetch performed for instruction cache misses.
- Disabling pipeline mode for data.
- Decreasing the maximum number of pending stores.
- Suppressing data fetches while instruction fetches are outstanding (and vice versa). The disabling can be either partial or selective.
- Artificially increasing network load by making two copies of each request and discarding one result. The difference between the arrival times of the responses to the requests can also be characterized.
- Approximating the behavior of a larger machine by introducing a shift register as a delay line from the network responses, thus increasing the latency of requests as seen by the processor. This can also be done selectively for different request types to measure the effect of latency on each individually.

2.3.3. Switch Instrumentation

The only configurability in the network is the ability to disable combining selectively for each switch. However, each 4×4 switch node will contain a XILINX chip dedicated to measurements, so extensive data collection can be performed. The processor will not be able to access the collected data directly; instead a PC will be used to download these data from the XILINX.

The most basic measurements that will be made on the network count the number of messages and packets passing through each node and the number of combines peformed at each node. Significant additional information can be obtained by tracking the inputs and outputs to the switch chips. Specifically, we can determine the number of requests in each queue at all times and thus produce histograms of queue lengths and whether the queue was blocked at that time (the queue is blocked based on the presence of a packet in a particular position, and not simply on the number of packets in the queue).

For the forward path, we also detect when a combine has occurred by observing when data is being sent to the wait buffer. For the return path, we obtain the corresponding information from a ‘‘decombine’’ signal, brought out of the chip just for the purpose of this measurement. In addition, by observing the decombine bit along with noting when data is sent from the forward path component to the wait buffer, we can accumulate similar statistics on the number of items in the wait buffer.

One useful piece of data that we do not expect to be able to obtain is the percentage of time that a request spends in the forward path compared to the return path (the amount of time spent in the MM is

constant). This data is available via simulation, however, and it can be validated by the data on total latency and queue length distributions that we will have.

2.3.4. MM Instrumentation

No logic is present in the MM for any performance monitoring. However, most quantities of interest that would be measured by the MM can instead be measured by the PE and network.

There is only a limited capability to change the configuration of the memory because its logic is implemented using multiple PALs rather than using XILINX chips. However, we do have the capability to increase the number of cycles it takes the memory to process a request. Also, we can vary the interleaving of the memory from full interleaving to no interleaving at all, although this change is actually made on the PEs and I/O processor.

2.4. Evaluation Strategy

Our evaluation strategies build on areas where our group has done previous research. We will address the specific value of unique features of the Ultracomputer architecture, particularly the impact of combining (Section 4.1). We will do basic measurements on speed of execution and parallel speedup on a variety of scientific applications (Section 4.2). As part of our on-going effort in the design of parallel coordination primitives and operating systems, we will investigate the performance of various interprocess coordination algorithms under a variety of conditions (Section 4.3).

These application and algorithm studies will be instrumented as described in Section 3, generating a wealth of measurement data useful for studying general issues in parallel computer design. In particular we are interested in investigating the interaction between the PE and the network, specifically latency-hiding changes in PE architecture (Section 4.4) and the effects of latency and contention in the network and at memory (Section 4.5). Many of the algorithms and applications we run will have access patterns that are data-dependent. We are interested in seeing the network behavior of these programs, which are, in general, not suitable for SIMD systems.

Mathematical modeling of software and hardware behavior will be a key part of all our investigations (Section 4.6). Verified models of the dependence of speed-up and processor utilization on application and architectural parameters are essential for accurate prediction of the behavior of future systems with much greater parallelism.

2.4.1. The Impact of Combining

Most of our experiments will be done with combining both enabled and disabled. Although a 16 PE machine may not be large enough to show significant performance degradation due to hot spot contention on the macroscopic level, the effect of combining in alleviating network saturation should be measurable. The 256-PE machine, if it becomes available, is expected to exhibit significant combining in most applications.

The effect of combining in only part of the network is of interest as a low cost alternative to systems where every node contains a full combining switch. The ability to disable combining selectively on a switch by switch basis will allow us to experiment with combining only in a single stage or on a tree to a particular memory module.

2.4.2. Scientific Applications

The Ultracomputer architecture's ease of programmability has attracted scientific application programmers and a variety of scientific codes have been ported to our prototypes. We have modified C and FORTRAN compilers to support parallel user programs. The extended languages permit shared variables, parallel loops, and parallel blocks. In addition, various program libraries are available, which contain coordination routines such as barrier synchronization and counting semaphores, routines for concurrent access to important data structures such as queues, and other items as well.

Previous work on the bus-based 8-PE Ultra II [2] showed a speedup limited by bus contention to around seven on many highly-parallel applications; the interconnection network on Ultra III should

overcome this limitation (naturally algorithms with significant serial portions do not perform this well). A first step will be to duplicate this research and see what speedups are obtained.

Previous research on scientific codes [11, 12] has included analyses of the types of operations that can be executed efficiently on various parallel architectures, identification of the operations used in current codes, and modification or redesign of the algorithms used in these programs to take better advantage of the architectures. Mathematical analysis of the algorithms themselves (e.g., their convergence properties and stability) has also been crucial in understanding what modifications can be made to improve speed without destroying other desirable features. The study of actual parallel implementations of currently used algorithms is an important check on theory and a necessary first step in the development of any new algorithm designed to take better advantage of parallelism.

Numerical codes have been written that serve as good test programs for determining optimal synchronization strategies. These codes can be used in the study of static versus dynamic synchronization techniques under different operating conditions (e.g., timesharing vs. single user mode).

Benchmarks like SPLASH [24] are also of interest, especially since they allow the comparison of our work with measurements on other architectures. We are currently in the process of porting the SPLASH benchmarks to the Ultra III prototype.

2.4.3. Synchronization algorithms and operating systems

The performance of the operating system is fundamentally dependent on the performance of the algorithms used for interprocess coordination and memory allocation. Knowledge gained from the tuning and measurement of these algorithms is invaluable in the structuring of more complex operating systems and application software.

We have developed a large set of algorithms and methodologies for interprocess coordination that utilize fetch-and-add, other fetch-and-ϕ primitives, and reflect (a mechanism to transmit an interrupt request from one PE, through the network and an MM to another PE). These methods are ‘‘bottleneck free’’ in the sense that they do not impose any serialization among processors beyond that required by the problem definition. Many of these algorithms have the additional desirable property of requiring an executing processor to issue no more than a constant number of instructions (independent of the degree of parallelism) in the absence of contention. For example, our fetch-and-add reader-writer algorithms have the property that, during periods when no writers are active, only a constant (and very small) number of instructions, including a single shared memory access, are needed to grant a reader lock.

Although these algorithms are bottleneck-free on an idealized PRAM, their actual performance is highly dependent on the target architecture and the run-time environment provided by the operating system. For example, although the BBN Butterfly [3] supports atomic fetch-and-add operations, they are slow and serialized on that architecture. Such systems impose an architectural bottleneck on bottleneck-free coordination algorithms based on fetch-and-add. In contrast, we anticipate that combining in the Ultracomputer network will eliminate this bottleneck, though we may discover other classes of network contention that affect the performance of coordination algorithms.

We experienced a related effect while experimenting with a BBN TC-2000 Butterfly at LLNL. The coordination algorithms in [20] make extensive use of the processor-local shared memory provided on the Butterfly to reduce network load. Their idea is to partition data structures so that waiting processors spin on local variables, which are updated by remote processors when the waited-for event has occurred. We investigated ‘‘triggers’’, which are a single bit of processor-local shared memory per processor. A processor spins on its local trigger; when it is set, the processor examines (remote) shared memory to determine the necessary actions. We modified the algorithms in [20] to use these triggers and timed the original and modified versions. The result was that both versions performed much worse than indicated in [20]. Clearly, these algorithms are very sensitive to the execution environment (the environments at LLNL and Argonne, where the referenced work was done, are different). We plan to repeat the above experment on our prototype, using the ‘‘reflect’’ network operation to provide the triggers.

The synchronization algorithms to be investigated include numerous competing implementations of binary and counting semaphores, readers/writers, and barriers for groups with both fixed and variable

sizes [8]. Differing implementations of the same primitive vary in memory requirements, access patterns, and fairness properties. We are interested in exploring the effectiveness of these different schemes. The behavior of these algorithms will be studied both by observing gross parameters such as execution time and speedup and by examining the detailed behavior of the system components during execution.

Although the Ultracomputer network is designed to minimize the impact of hot-spot traffic, we remain interested in minimizing network load due to coordination. Therefore, our experiments will study the impact of mechanisms that adjust shared memory polling frequency on both the performance of inter-processor coordination and unrelated system activity. These adjustments may be a function of waiting time or indications that the awaited resource may soon become available. Finally, we will use the Ultra III prototype to explore a new class of coordination algorithms that use each processor's reflect register as a single bit of processor-local shared memory for spin-waiting [7]. This class of algorithms may generate substantially less network traffic than those that utilize conventional shared variables.

The communication algorithms that will be investigated include implementations of several families of queues, multi-queues and an asynchronous signal delivery system [5]. The queue algorithms primarily differ in the relative ordering of inserts and deletes, their ability to support internal deletions, and the mechanisms utilized to store and access individual items. Results from experiments on queues will be useful not only for the selection and tuning of these queue algorithms but also for the choice of data storage mechanisms for other applications.

Several algorithms have been proposed for parallel memory allocation on Ultracomputers [26, 6]. These algorithms include parallel buddy and first-fit systems. Our evaluation of these algorithms will consider fragmentation as an additional performance characteristic.

Our Symunix operating system kernel, a heavy user of the kinds of algorithms and data structures discussed in this section, has two internal features designed to help with their evaluation. First, extensive software instrumentation includes event counting and two kinds of histograms. For example, every synchronization delay is fully histogrammed to show the distribution of waiting times. Second, several basic highly parallel algorithms can be replaced with other variants, including less parallel or uniprocessor versions, allowing us to measure the tradeoffs between the serial and parallel versions.

Some Symunix experiments were performed on our old Ultra II prototype to compare the performance effects of fetch-and-add algorithms to semaphores [5]. In one experiment, we measured the number of open and close operations executed per second by up to eight (the number of PEs in our system) processes using three different Unix kernels: a uniprocessor kernel, Symunix with the fetch-and-add algorithms replaced by semaphores, and normal Symunix.

The results were that normal Symunix showed near-linear speedups until the effect of the bus in the Ultra II prototype became apparent at around six processes. The uniprocessor kernel showed a constant number of operations per second independent of the number of processes, and, as expected, replacing the primitives with ones using semaphores performed worst. In the latter case, not only was the achievable rate at any given number of processes the lowest, but it actually **decreased** slightly as processes were added.

We will also have the capability to conduct experiments on classic questions of load balancing, locality management and cache behavior in the context of parallel systems.

2.4.4. Processing Element Design

One of the areas we will be concentrating on is the affect of network behavior on the processor. To this end, we will both perform measurements and modify the behavior of the processor. The central measurment in this investigation is *processor utilization*. As discussed in Section 2.1, the Am29050 does not necessarily stall when memory requests are pending. Therefore we measure utilization by counting the number of cycles the processor spends in ''pipeline hold'' mode, which can be decoded from three status lines provided by the Am29050. This measurement will be made while running the various applications and algorithms described above. With this baseline data, the investigation will proceed in two areas:

First, we will characterize the reasons that stalls can occur and count the number of stall cycles due to each. For example, stalls can occur if a request occurs at wrong cycle parity (the network requires an

even number, possibly zero, of cycles between messages). Stalls will also occur if a store is done and this PE already has eight outstanding stores in the network. The largest number of stall cycles are expected to be caused by network and memory latency. Finally, stalls will also occur due to ‘‘fence’’ operations, which enforce sequential (or weaker) consistency models. Within these categories, the stalls can be due to either data or instruction accesses. These accesses, in turn, can occur either because the item being loaded is non-cachable, or because a cache miss occurred. Each of these cases is detectable and can thus be counted.

Second, we will vary the processor parameters described in Section 3.2.2, such as the number of stores allowed to be pending simultaneously, and see how this changes the gross performance (CPU time and speedup), the processor utilization, and the detailed breakdown of types of stall cycles mentioned above. Again, these experiments will be performed on various types of applications and algorithms. This information will enable us to determine the importance of each of the latency-reducing measures we have provided in the PE.

Subsequent research in this area may involve further increasing the number of outstanding loads by providing a mechanism for the processor to fetch data into registers internal to the XILINX and only later to request such data. This is similar to the mechanism studied under simulations in [1].

The information we obtain from these investigations will assist in the design of algorithms and compiler optimization, as well as the design of future PEs for larger systems.

2.4.5. The Behavior of Multistage Interconnection Networks

Other experimental systems with good performance monitoring capabilities have mesh or hypercube connections (e.g., Alewife [27], the Message-Driven Processor [30], Hnet [25], and the Stanford Dash [28]), or use an unbuffered multi-stage interconnection network (e.g., the Transit Network [14] or the BBN Butterfly [3]). Buffered multistage interconnection networks have been the subject of many analytical and simulation studies, but the only opportunity for measurement has been the RP3 [29]. The RP3 implementation did not include combining, so Ultra III will offer the first opportunity to measure its effect. Since the Ultra III prototype operates over seven orders of magnitude faster than detailed network simulations, the quantity and detail of data that can be gathered would be prohibitively expensive to obtain under simulation.

We will be able to measure the number of combines that occur, throughput of each node, and various statistics relating to queue lengths. Statistics can be gathered during the course of other work on applications and operating systems as well as by applying artificial stimuli to the network. Such stimuli will include synthetic applications similar to the MAD kernels [21] as well as special XILINX firmware designed to stress the network.

Previous simulation studies of network behavior [23, 19] have shown combining to be effective in alleviating tree saturation due to hot spots at memory. Using statistics obtained with both real and artificial loading of the network, we plan to:

- Verify simulation and analytical models of single hotspot behavior.
- Determine the behavior of the network under multiple hot spots. We are interested in how many hot spots are typically active simultaneously and what congestion patterns appear in the network with several active hot spots.
- Observe queueing delays on the return from memory. Simulations under stochastic loads from the PEs have shown these delays to be less than would be expected if the memories were randomly issuing responses into the network.
- Characterize traffic levels and hotspot concentrations at which combining is useful and effective. More than two-way combining may be required for large networks [17]; measuring the amount of increased combining in the earlier stages due to blocking at the later stages will give us better estimates as to how large a network must be before greater than two-way combining is needed.

2.4.6. Mathematical Models of System Behavior

We are interested in developing models of parallel system behavior that can be extrapolated to larger systems. Our previous work in system modeling has concentrated on network behavior [22, 19] and on the behavior of combining switches [4] under stochastic loads, sometimes with a fixed percentage of hot spot traffic.

Memory traffic generated from real applications may be significantly different from that assumed in these stochastic models. The traffic is likely to be more bursty, and the traffic intensity of messages issued by a PE will go down if the PE must wait too long for responses from memory. We would like to characterize the memory access behavior of different types of applications, with particular emphasis on the feedback effect of contention and latency (the higher the traffic, the more contention, therefore, the longer the wait; but the longer the wait, the less traffic issued, therefore, the less contention). This effect cannot be modeled exactly with either open network models, such as those used in [18, 15], which allow a processor to issue an indefinite number of requests, or with simple closed network models, such as [13], in which a processor has a fixed number of messages in the system independent of the round-trip latency of a single message. Relevant measurements would include a histogram of access to different memory modules, the relationship of average network latency to processor utilization and speedup, the results of different interleaving strategies, and the results of adding extra latency to network accesses.

The models of memory access behavior can then be related to hardware models of architectural features that can be used to hide latency, like local memory and processor pre-fetch capability, and to those that can be used to lessen contention, like buffers for higher network bandwidth and combining to reduce hotspot contention. We hope to provide a model that will correlate the microscopic behavior of the system (such as processor utilization, number of outstanding requests by the PEs, and queue sizes in the network) with the macroscopic behavior of the system (execution times and speedups).

2.5. Results from Prior NSF Support

Since its inception in 1979, the NYU Ultracomputer project has been continually supported by the DOE. From 1985 through 1988, substantial support was received from IBM and DARPA for our work with the RP3 project, which initially emphasized software (primarily OS) development and subsequently was expanded to also provide support for RP3-specific combining switch VLSI design. Over the last 13 years, NSF provided approximately 20% of the Ultracomputer project support. Our two most recent NSF grants were ‘‘VLSI Architecture for Ultraparallel Computers’’ (DCR-8413359), for $840,000 during the period from April 1, 1985 to September 30, 1988, and ‘‘Ultra III: Implementing a Scalable Shared-Memory Multiprocessor’’ (MIP-8915488), for $765,000, during the period March 1, 1990 through the present. The major efforts of these two grants were directed towards the design and construction of the Ultra III prototype, as described in Section 2 with the earlier grant supporting the research that led to the systolic queue design used in our VLSI chips and the latter project actually constructing the prototype.

Prior to this work, we previously constructed several bus-based Ultracomputer prototypes, called Ultra II, each containing eight PEs and four MMs. All MMs supported fetch-and-add; some newer ones (designed in support of the Ultra III prototype) supported other fetch-and-ϕs as well. Each PE contained a 10 MHz Motorola MC68010 augmented with Weitek 1164/65 floating-point units, a 32KB two-way set-associative cache, hardware support for fetch-and-add, two serial ports, and a small amount of RAM and ROM. The PEs and MMs were placed on a single ‘‘Ultrabus’’ that arbitrated among the PEs. A second port on each MM was connected to a PDP-11 that, in addition to serving as an all-purpose I/O controller, connected the prototype to the NYU network, and hence to the Internet as well.

The Ultra II prototypes were in routine use for four years for software development, algorithms and application studies, and scientific production, both within and outside NYU. The computer science department has offered several courses on programming MIMD shared-memory computers for which students were required to develop, run, and analyze programs on the prototypes. These courses have been valuable in disseminating the methodologies of programming the Ultracomputer in C, FORTRAN, and Lisp for a wide variety of applications. Thus, the reliability of our hardware and software has been confirmed. Over the last few years, the Ultra II prototypes have not seen much use. However, we do keep one prototype running. We expect the Ultra III prototype hardware and software to be as reliable as

our Ultra II prototypes have been.

We have recently constructed a 4-PE system using all the component types required for the Ultra III prototype. All that remains is to replicate our PE and MM boards and to have NCR, one of our industrial partners, re-fabricate our switches using a more advanced process and higher pin-count packages than those available to us through MOSIS (the current, fully-functional 4×4 network uses MOSIS-fabricated chips). We expect the full system to be completed during in the first three months of 1993.

The PE to MM interconnection network for Ultra III is composed of custom VLSI chips which were designed by our group and have been fabricated by MOSIS in 2μ CMOS. We currently use two part types: a forward path component, which routes and combines messages from the PEs to the MMs, and a return path component, which routes responses from the MMs to the PEs and decombines messages that have been combined on the forward path. The return path component contains a associative wait buffer that stores the combined requests until the response arrives from memory. Using MOSIS 132-pin packages, four of each component are required for a 2×2 switch; using 208-pin packages to be provided by NCR, two of each component are required.

The Ultracomputer group has also accomplished the following:

- Discovered the fetch-and-add primitive and produced numerous coordination algorithms using it. This work has been, and continues to be, central to our efforts.
- Performed extensive simulations on the PE to MM interconnection network.
- Produced analytic models for the behavior of the switches in our combining network.
- Designed a fully-symmetric version of Unix (one version has been implemented; the second is under development).
- Produced a parallel debugger.
- Ran many scientific applications on our prototypes and investigated the parallelizability of numerical applications.
- Performed software and VLSI development specific to the RP3 for IBM.

The Ultracomputer Research Laboratory is one of the major research projects in the Computer Science Department in NYU and there has been substantial student involvement in our project. Shortly after the Ultra II prototype was completed, a number of courses were offered in parallel programming techniques using that system; we anticipate offering similar courses using the Ultra III prototype when it is completed.

A number of PhD theses have been supported by our project. Six have been completed:

- Isaac Dimitrovsky, on a parallel LISP system
- Clyde Kruskal, on analysis of parallel algorithms
- Yue-Sheng Liu, on network simulation
- Lawrence Rudolph, on software structures for parallel computing
- Patricia Teller, on TLB consistency mechanisms
- Jim Wilson, on parallel queue algorithms

Two theses are being completed:

- Susan Dickey, on the design of combining switches.
- Jan Edler, on parallel operating systems

Other graduate students who have worked on our project are Brian Anderson, Wayne Berke, Guoying Chen, Susan Flynn, Eric Freudenthal, Benchiao Jai, Larry Kaplan, Richard Kenner, Susan Lambe, Mikha Liberman, Markantonatos Mitsolides, Nagabhu Prabhu, Debera Rennels, Edith Schonberg, Milan Tuba, John Turek, Jiarui Wang, Cheuk-Bun Wong, David Wood, Chin Woo You, and Yongtao You.

One NYU undergraduate student, Miriam Ayala, has also worked on the project, as have three undergraduates visiting from France.

Some of the students listed above have later joined our project as full-time staff members.

Since the inception of our project, we have produced almost 300 reports, most of which are available in both electronic and paper form. As of now, we have written 180 Ultracomputer Notes, two Ultracomputer Application Notes, seven Ultracomputer Documentation Notes, 28 Ultracomputer Hardware

Notes, two Ultracomputer Scientific Programming Notes, and 66 Ultracomputer System Software Notes.

The following 35 publications have appeared within the past five years:

George Almasi and Allan Gottlieb, *Highly Parallel Computing*, Benjamin/Cummings Publishing Company, Redwood City, Calif., 1989, 519 pages. Second edition under preparation.

Yosi Ben-Asher, David Egozi, and Assaf Schuster, ''SIMD Algorithms for 2-D Arrays in Shuffle Networks'', *Proc. 16th Annual Intern. Symp. on Comp. Arch.*, 1989.

Richard Cole and Uzi Vishkin, ''Approximate Parallel Scheduling. Part I: The Basic Technique with Applications to Optimal Parallel List Ranking in Logarithmic Time'', *SIAM J. on Comp.* **17** No. 1, February 1988, pp. 128-142.

Richard Cole and Uzi Vishkin, ''The Accelerated Centroid Decomposition Technique for Optimal Parallel Evaluation in Logarithmic Time'', *Algorithmica*, 1988 (3), pp. 329-346.

Richard Cole and Uzi Vishkin, ''Approximate Parallel Scheduling. Part II: Applications to Optimal Parallel Graph Algorithms in Logarithmic Time'', *Information and Computations* **92** No. 1, 1991, pp. 1-47.

Richard Cole and Uzi Vishkin, ''Faster Optimal Parallel Prefix Sums and List Ranking'', *Information and Computations* **81** No. 3, 1989, pp. 334-352.

Susan Dickey, Allan Gottlieb and Richard Kenner, ''Using VLSI to Reduce Serialization and Memory Traffic in Shared Memory Parallel Computers'', *Advanced Research in VLSI: Proc. Fourth MIT Conference*, Charles E. Leiserson, ed, April 1986, pp. 299-316.

Susan Dickey, Allan Gottlieb, Richard Kenner and Yue-Sheng Liu, ''Designing VLSI Network Nodes to Reduce Memory Traffic in a Shared Memory Parallel Computer'', *Circuits Systems Signal Process* **6** No. 2, 1987, pp. 217-38.

Susan Dickey and Yue Sheng Liu, ''Simulation and Analysis of Enhanced Switch Architectures for Interconnection Networks in Massively Parallel Shared Memory Machines'', *Proc. 2nd Symposium on the Frontiers of Massively Parallel Computation*, McLean, Virginia, October 1988, pp. 487-490.

Susan Dickey and Richard Kenner, ''Using Qualified Clocks in the NORA Clocking Methodology to Implement a Systolic Queue Design'', *Proc. Brown/MIT Conference on Advanced Research in VLSI*, March 1992, pp. 165-179.

Susan Dickey and Richard Kenner, ''Hardware Combining and Scalability'', *Proc. 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, San Diego, June 29 - July 1, 1992, pp. 296-305.

Susan Dickey and Richard Kenner, ''Combining Switches for the NYU Ultracomputer'', *Proc. Fourth Symposium on the Frontiers of Massively Parallel Computation*, McLean, Virginia, Oct 19-21, 1992.

Susan Dickey and Ora E. Percus, ''Performance Differences Among Combining Switch Architectures'', *Proc. 21st International Conference on Parallel Processing* **I** August, 1992, pp. 110-117.

Susan Dickey and Ora E. Percus, ''Performance Analysis of Clock-Regulated Queues with Output Multiplexing in Three Different Types of 2 by 2 Crossbar Switch Architectures'', *Journal of Parallel and Distributed Computing* **16**, 1992, pp. 27-40.

Anne Dinning and Edith Schonberg, ''An Empirical Comparison of Monitoring Algorithms for Access Anomaly Detection'', *Proc. 2nd ACM Symposium on Parallel Programming*, March 1990.

Jan Edler, Jim Lipkis, and Edith Schonberg, ''Process Management for Highly Parallel UNIX Systems'', *Proc. Usenix Workshop on UNIX and Supercomputers*, Pittsburgh, 1988, pp. 1-17.

Jan Edler, Jim Lipkis, and Edith Schonberg, ''Memory Management in Symunix II: A Design for Large-Scale Shared Memory Multiprocessors'', *Proc. Usenix Workshop on UNIX and Supercomputers*, Pittsburgh, 1988, pp. 151-168.

Eric Freudenthal and Allan Gottlieb, ''Process Coordination with Fetch-and-Increment'', *Proc. ASPLOS IV*, Santa Clara, CA, April 1991, pp. 260-268.

Allan Gottlieb, ‘‘An Overview of the NYU Ultracomputer Project’’, in *Experimental Parallel Computing Architectures*, Jack J. Dongarra, ed., North Holland, 1987, pp. 25-95.

Allan Gottlieb, ‘‘Scalability, Combining, and the NYU Ultracomputer’’, *Proc. 1990 Parallel Computing Workshop*, Ohio State University, March 1990.

Allan Gottlieb, ‘‘An Outsider’s View of Dataflow’’ in *Advanced Topics in Data-flow Computing*, Lubomir Bic and Jean-Luc Gaudiot, ed., Prentice Hall, 1991, pp. 573-581.

Allan Gottlieb, ‘‘Architectures for Parallel Supercomputing’’, *Proc. PACTA ’92 (Parallel Computing and Transputer Applications*, Barcelona, September 1992, pp. 39-47.

Torbjörn Granlund and Richard Kenner, ‘‘Eliminating Branches using a Superoptimizer and the GNU C Compiler’’, *Proc. SIGPLAN ’92 Conference on Programming Language Design and Implementation* San Francisco, June 17-19, 1992, pp. 341-352.

Malvin Kalos and Ora E. Percus, ‘‘Random Number Generation for MIMD Parallel Processors’’, *Journal of Parallel and Distributed Computing*, **6**, 1989, pp. 477-497.

Richard Kenner, Ronald Bianchini, Susan Dickey, and Patricia J. Teller, ‘‘A Compact Design for a Highly-Parallel Shared-Memory MIMD Computer’’, *Proc. CompCon Spring ’89* San Francisco, Feb.-Mar. 1989, pp. 264-269.

Richard Kenner, Susan Dickey, and Patricia J. Teller, ‘‘The Design of Processing Elements on a Multiprocessor System with a High-Bandwidth, High-Latency Interconnection Network’’, *Proc. 22nd HICSS* **I** Kailua-Kona, Jan. 1989, pp. 319-328.

Gad M. Landau and Uzi Vishkin, ‘‘Fast Parallel and Serial Approximate String Matching’’, *Journal of Algorithms* **20** No. 2, 1989, pp. 262-272.

Yue-Sheng Liu and Susan Dickey, ‘‘Simulation and Analysis of Enhanced Switch Architectures for Interconnection Networks in Massively Parallel Shared Memory Machines’’, *Proc. 2nd Symp. on the Frontiers of Massively Parallel Computation*, Oct. 1988.

Ora E. Percus and Jerome K. Percus ‘‘Long Range Collelations in Linear Congruential Generators’’, *J. Comp. Phys.*, **77**, 1988, pp. 267-270.

Ora E. Percus and Jerome K. Percus ‘‘Some Results Concerning Clock-Regulated Queues’’, *Computer Architecture News*, **16(4)** 1988, pp. 138-144.

Ora E. Percus and Jerome K. Percus ‘‘Models for Queue Length in Clocked Queueing Networks’’, *Communications of Pure and Applied Mathematics*, **XLIII**, 1990, pp. 273-289.

Ora E. Percus and Jerome K. Percus ‘‘Queue Length Distribution in a Markov Model of a Multistage Clocked Queueing Network’’, *Communications and Pure and Applied Mathematics* **43**, 1990, pp. 685-693.

Ora E. Percus and Jerome K. Percus, ‘‘Elementary Properties of Clock-regulated Queues’’, *SIAM J. on Appl. Math* **50** No. 4, August 1990, pp. 1166-1175.

Ora E. Percus and Jerome K. Percus ‘‘Time Series Transformations in Clocked Queuing Networks’’, *Communications on Pure and Applied Mathematics* **44** No. 9, 1991.

Patricia J. Teller, Richard Kenner and Marc Snir, ‘‘TLB Consistency on Highly-Parallel Shared-Memory Multiprocessors’’, *Proc. 21st HICSS* **I** Kailua-Kona, Jan. 1988 pp. 184-193.

3. Bibliography

[1] William Y. Chen, Scott A. Mahlke, and Wen-mei W. Hwu, ‘‘Tolerating First Level Memory Access Latency in High-Performance Systems’’, *Proc. 21st Annual ICPP* **I**, pp. 36-43 (August 1992).

[2] Jerome Chiabaut, Vladimir Fleyshgakker, and Anne Greenbaum, ‘‘Porting Scientific Applications to the NYU Ultracomputer’’, Ultracomputer Note 144, Courant Institute, NYU (1988).

[3] W. Crowther, J. Goodhue, R. Gurwitz, R. Rettberg, and R. Thomas, ‘‘The Butterfly (TM) Parallel Processor’’, *IEEE Computer Architecture Technical Committee Newsletter*, pp. 18-45 (September 1985).

[4] Susan R. Dickey and Ora E. Percus, ‘‘Performance Differences among Combining Switch Architectures’’, *Proc. 21st Annual ICPP* **I**, pp. 110-117 (August 1992).

[5] Jan Edler, ‘‘Practical Structures for Parallel Operating Systems’’, Ph.D. Thesis, Courant Institute, NYU (1993).

[6] Jan Edler, Jim Lipkis, and Edith Schonberg, ‘‘Memory Management in Symunix II: A Design for Large-Scale Shared Memory Multiprocessors’’, *Proc. USENIX Workshop on UNIX and Supercomputers* (September 1988). Also available as Ultracomputer Note 135, Courant Institute, NYU.

[7] Eric Freudenthal, ‘‘Evaluation of Interprocessor Triggers as a Coordination Primitive for Shared-Memory MIMD Computers’’, 1992 Massively Parallel Computing Initiative Yearly Report, Lawrence Livermore National Labs ().

[8] Eric Freudenthal and Allan Gottlieb, ‘‘Process Coordination with Fetch-and-Increment’’, *Proc. ASPLOS-IV*, pp. 260-268 (April 1991).

[9] G. R. Goke and G. J. Lipovski, ‘‘Banyan Networks for Partitioning Multiprocessor Systems’’, *Proc. 1st Annual International Symposium on Computer Architecture*, pp. 21-28 (1973).

[10] Allan Gottlieb, Ralph Grishman, Clyde P. Kruskal, Kevin P. McAuliffe, Larry Rudolph, and Mark Snir, ‘‘The NYU Ultracomputer—Designing a MIMD, Shared-Memory Parallel Machine’’, *IEEE Transactions on Computers* **32** (2), pp. 175-189 (February 1983).

[11] Anne Greenbaum, ‘‘Solving Sparse Triangular Linear Systems Using FORTRAN with Parallel Extensions on the NYU Ultracomputer Prototype’’, Ultracomputer Note 99, Courant Institute, NYU (1986).

[12] Anne Greenbaum, Congming Li, and Han Zheng Chao, ‘‘Comparison of Linear System Solvers Applied to Diffusion-Type Finite Element Equations’’, Ultracomputer Note 126, Courant Institute, NYU (1987).

[13] Hong Jiang and Laxmi N. Bhuyan, ‘‘MVAMIN: Mean Value Analysis Algorithms for Multistage Interconnection Networks’’, *Journal of Parallel and Distributed Computing* **12**, pp. 189-201 (1991).

[14] Thomas F. Knight, Jr., ‘‘Technologies for Low-Latency Interconnection Switches’’, *Symposium on Parallel Algorithms and Architectures*, (June 1989).

[15] Clyde P. Kruskal, Marc Snir, and Allan J. Weiss, ‘‘On the Distribution of Delays in Buffered Multistage Interconnection Networks for Uniform and Nonuniform Traffic’’, *Proc. 1984 International Conference on Parallel Processing*, pp. 219-220 (1984).

[16] Duncan H. Lawrie, ‘‘Access and Alignment of Data in an Array Processor’’, *IEEE Transactions on Computers* **24** (12), pp. 1145-1155 (December 1975).

[17] Gyung Ho Lee, Clyde P. Kruskal, and David J. Kuck, ‘‘The Effectiveness of Combining in Shared-Memory Parallel Computers in the Presence of Hot-Spots’’, *Proc. 1986 International Conference on Parallel Processing*, pp. 35-41 (August 1986).

[18] T. Lin and L. Kleinrock, ‘‘Performance Anlysis of Finite-Buffered Multistage Interconnection Networks with a General Traffic Pattern’’, *Proc. ACM Sigmetrics Performance* **91**, pp. 68-78

(1991).

[19] Yue-Sheng Liu, ‘‘Architecture and Performance of Processor-Memory Interconnection Networks for MIMD Shared Memory Parallel Processing Systems’’, Ph. D. Dissertation, New York University, Department of Computer Science (September 1990).

[20] John M. Mellor-Crummey and Michael L. Scott, ‘‘Algorithms for Scalable Synchronization on Shared-Memory Multiprocessors’’, *ACM TOCS* **9** (1) (February 1991).

[21] Arun Nanda and Lionel M. Ni, ‘‘MAD Kernels: An Experimental Testbed to Study Multiprocessor Memory System Behavior’’, *Proc. 21st Annual ICPP* **I**, pp. 28-35 (August 1992).

[22] Ora E. Percus and Jerome K. Percus, ‘‘Elementary Properties of Clock-regulated Queues’’, *SIAM Journal on Applied Mathematics* **50**, pp. 1166-1175 (1990).

[23] Gregory F. Pfister and V. Alan Norton, ‘‘‘Hot Spot’ Contention and Combining in Multistage Interconnection Networks’’, *IEEE Transactions on Computers* **34** (10), pp. 943-948 (October 1985).

[24] Jaswinder Pal Singh, Wolf-Dietrich Weber, and Anoop Gupta, ‘‘SPLASH: Stanford Parallel Applications for Shared-Memory’’, *Proc. 21st Annual ICPP* **I**, pp. 36-43 (August 1992).

[25] David Smitley, Frank Hady, and Dan Burris, ‘‘Hnet: A High-performance Network Evaluation Testbed’’, *Proc. 21st Annual ICPP* **I**, pp. 276-279 (August 1992).

[26] James Wilson, ‘‘Operating System Data Structures for Shared-Memory MIMD Machines with Fetch-and-Add’’, Ph.D. thesis, Courant Institute, NYU (1988).

[27] Anant Agarwal, et. al, ‘‘The MIT Alewife Machine: A Large-Scale Distributed-Memory Multiprocessor’’, Technical Report MIT/LCS/TM-454, MIT Laboratory for Computer Science (June 1991).

[28] Daniel Lenoski et. al, ‘‘The Stanford Dash Multiprocessor’’, *IEEE Computer* **25** (3) (March 1992).

[29] G. Pfister, et. al, ‘‘The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture’’, *Proc. 1985 International Conference on Parallel Processing*, pp. 764-771 (August, 1985).

[30] William J. Dally et. al, ‘‘Design and Implementation of the Message-Driven Processor’’, *Proceedings of the 1992 Brown/MIT Conference on Advanced Research in VLSI and Parallel Systems*, pp. 5-24 (March 1992).

4. Biographical Sketches

Allan Gottlieb
Susan Dickey
Jan Edler
Eric Freudenthal
Anne Greenbaum
Richard Kenner
Ora Percus

Table of Contents

1. Project Summary 1
2. Project Description 2
2.1. Introduction 2
2.2. Ultra III Design 3
2.2.1. Processing Element 3
2.2.2. MM Design 5
2.2.3. Interconnection Network 5
2.3. Instrumentation Tools 6
2.3.1. External Measurements 6
2.3.2. PE Instrumentation 6
2.3.2.1. PE Measurements 6
2.3.2.2. PE Configurability 7
2.3.3. Switch Instrumentation 7
2.3.4. MM Instrumentation 8
2.4. Evaluation Strategy 8
2.4.1. The Impact of Combining 8
2.4.2. Scientific Applications 8
2.4.3. Synchronization algorithms and operating systems 9
2.4.4. Processing Element Design 10
2.4.5. The Behavior of Multistage Interconnection Networks 11
2.4.6. Mathematical Models of System Behavior 12
2.5. Results from Prior NSF Support 12
3. Bibliography 16
4. Biographical Sketches 18