A COMPACT DESIGN FOR A HIGHLY-PARALLEL SHARED-MEMORY MIMD COMPUTER

Richard Kenner, Ronald Bianchini, Susan Dickey, and Patricia J. Teller

Ultracomputer Research Laboratory
Courant Institute of Mathematical Sciences
New York University
251 Mercer Street
New York, NY 10012
(212) 998-3339

Ultracomputer Note #158
January, 1989

Abstract

The Ultracomputer architecture connects hundreds or possibly thousands of processing elements (PEs), each containing a cache but no local memory, to an equal number of memory modules (MMs) via an interconnection network constructed of custom VLSI components that combine (merge) requests from different PEs destined to the same memory location. The network provides a high-bandwidth path from the PEs to MMs but the memory latency is significantly larger than that encountered in either a uniprocessor or small bus-based multiprocessor. The PE must exploit the high available bandwidth and minimize the effect of network latency.

This paper presents a design of a 64-PE Ultracomputer prototype using AMD Am29000 CPUs, including a description of system packaging using a backplane-free technology. The prototype is expected to be operational in 1990.

1. Introduction

Large multifunction computers are a universal tool for science and engineering. Current supercomputers are approaching engineering limits in their uniprocessor designs; though component densities are increasing, raw clock speed improvements are not keeping pace. To achieve the much faster aggregate speeds needed in the future, parallel computation must be used. Commercial supercomputers are evolving in this direction; the top-of-the-line CRAY computers have four separate processors that can cooperate through shared memory.

Preventing performance degradation due to interprocessor communication is the central problem which any massively parallel computer architecture must solve. For many algorithms it is possible to define special-purpose machine architectures that permit high execution rates when using particular interprocessor connections. However, as the application is studied further, improved algorithms with different communication requirements may emerge that place the special-purpose computer at a severe disadvantage compared to more flexible machines. Another potential problem with special-purpose machines is that many applications are composite, comprising diverse algorithms having varied architectural requirements.

In addition to offering large speedups for sufficiently large problems, general-purpose machines are effective at providing simultaneous execution of a number of smaller problems. For example, a system with 1000 processors should not only be capable of delivering a nearly 1000 fold speedup when devoted to a single major computation, but should also permit concurrent execution of 20 more modest jobs achieving on each a speedup of approximately 50.

Although message-passing architectures are indeed quite attractive for distributed computing, for thousands of processors cooperating to solve a single large-scale scientific problem, high-speed concurrent access to shared memory is more effective. In particular, for applications in which substantial access to very large sets of data is needed and in which the patterns of access are spatially and temporally unpredictable, shared memory architectures should enjoy a significant advantage in efficiency and programmability.

In this paper we present the NYU Ultracomputer architecture (see also [6]), designed for efficient shared memory parallel computation, and describe the construction of an intermediate size 64-processor prototype. The first section presents the overall architecture and the

This work was supported in part by the Applied Mathematical Sciences Program of the U.S. Department of Energy under contract DE-AC02-76ER03077, in part by the National Science Foundation under grant DCR-8413359, and in part by I.B.M. under joint study agreement N00039-84-R-0605(Q).

characteristics of the interconnection network, which are crucial to all other aspects of the design. The remaining sections give implementation methods using current technology.

2. The Ultracomputer Architecture

The Ultracomputer architecture, shown in Figure 1, is designed to scale to thousands of PEs. Unlike an idealized multiprocessor, called a *paracomputer* by [16] and classified as a WRAM by [2], it does not assume single-cycle access to shared memory but instead connects the PEs to an equal number of MMs via a multistage interconnection network built from custom VLSI switches that merge simultaneous requests for the same memory location.

The processor's instruction set is augmented by fetch-and-ϕ operations (which can be defined for any associative function ϕ), each of which atomically modifies a memory location and returns its old value. For example, fetch-and-add (F&A) is an indivisible add to memory: F&A(X, e), where X is an integer variable and e is an integer expression, returns the (old) value of X and replaces X in memory with the sum $X + e$. Using fetch-and-ϕ, it is possible to write efficient parallel algorithms with reduced synchronization overhead. Early versions of such algorithms are given in [7]; see [18] for a large collection of such algorithms.

2.1 The Interconnection Network

The Ultracomputer network (shown in Figure 2 for eight PEs) has the topology of Lawrie's Ω network [11] and the SW Banyan of [5]. It consists of $\log N$ stages, where N is the number of PEs in the system. In this paper, we discuss a network with $\log_2 N$ stages, each having $N/2$ nodes, where a node is a 2×2 switch that routes requests from PEs to MMs and responses (to these requests) from MMs to PEs. A switch has four bidirectional ports, two connecting either to PEs or switches closer to the PEs and two connecting either to MMs or switches closer to the MMs. The switch is buffered in order to avoid the bandwidth limitation of unbuffered networks [9].

A switch consists of two independent routers, one for requests and the other for responses and can *combine*, or merge, two requests that refer to the same memory location and that are simultaneously present in its queues. Combining reduces network saturation resulting from *hot-*

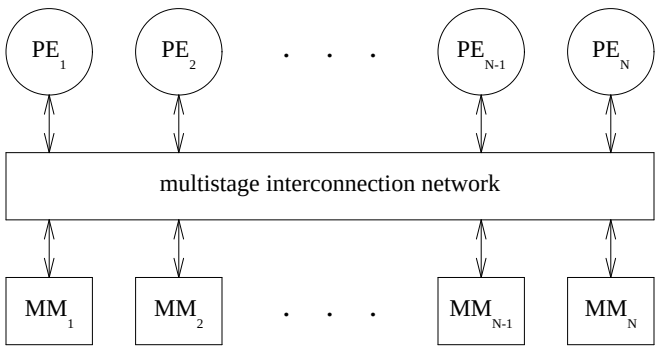

Figure 1. Ultracomputer Architecture

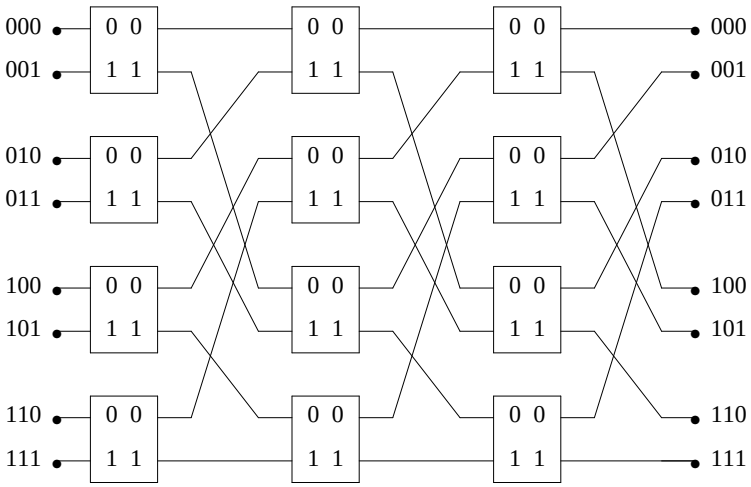

Figure 2. Ultracomputer Network (N = 8)

spot traffic [15], characterized by many concurrent requests to the same location. Since this is a common access pattern for parallel programs performing synchronization operations, combining is important for adequate system performance.

These switches can potentially combine any fetch-and-ϕ operation[1]. When two messages destined for the same location meet at a switch, one is saved in a *wait buffer* and a merged message is sent towards the MMs. When a switch recognizes that an MM response matches a message in its wait buffer, a second response is generated in order to satisfy both requests.

A 32-bit store requires at least 64-bit messages (32 for address and 32 for data). Options range from bit-serial transmission to 64-bit ports. Larger packet sizes produce higher network bandwidth but increase the difficulty of packaging the network. The combining logic is simplified by putting the address and operation code in the first packet; the systolic design of the queues requires that messages be composed of an even number of packets [3]. The first requirement imposes a minimum width; the second makes a larger width of limited utility. Therefore, 32-bit requests are transmitted as two packets; the first containing the address, the second the data. The packets of a message are pipelined: switches transmit each packet of a message as soon as possible rather than waiting for the entire message.

2.2 Network Performance

Overall system performance for very large numbers of processors critically depends on the message throughput that can be achieved by the interconnection network. Since a PE may sometimes require the data from a previous request before it can make another request, message throughput is limited not only by the theoretical bandwidth of the network but by its latency, the time from the generation of a request until a response is received.

Network latency has three components: the basic transit time through each network stage, the pipeline-fill time for the packets in a message after the first, and the waiting time in the queues

[1] *Loads* and *stores* are also fetch-and-ϕ operations. In a load, ϕ is the projection operation $\pi_1(x,y)=x$ so that fetch-and-$\pi_1(V,e)$ returns the value of V and sets V to $\pi_1(V,e)=V$, i.e. leaves V unchanged. Since e is not used, it need not be transmitted. Similarly, for a store, ϕ is $\pi_2(x,y)=y$.

at each stage. The minimum network transit time, in the absence of any delay due to routing conflicts, is thus equal to the sum, not the product, of the number of stages and the number of packets per message.

From [10] (see also [14]), assuming unbounded buffer size and uniform distribution of memory addresses, the expected value of the waiting time in the first stage is

$$t\left[\frac{m^2 p(1-1/km)}{2(1-mp)}\right]$$

and the waiting time at later stages is approximately

$$t(1 + \frac{4mp}{5k})\left[\frac{m^2 p(1-1/k)}{2(1-mp)}\right] ,$$

where t is the cycle time of the switch, m is the number of packets per message, k is the size of the crossbar, p is the offered load in messages per cycle, and the correction factor in the second formula is obtained from simulations (see also [13] for a similar correction factor). Note the quadratic term in m. It is known that even moderately large values of m have a large effect on the *variance* of the latency [9].

To determine the accuracy of these estimates with bounded buffer size and non-uniform traffic, we simulate networks composed of 2×2 switches under various assumptions of switch architecture, PE request generation, and memory behavior [4]. Although with infinite buffers the theoretical network bandwidth allows up to one packet per cycle per port, with finite buffers the *effective throughput* (the number of messages per cycle actually sent) will be less than that due to blocking. Various values of buffer size and *offered load,* (the probability of generating a request on any cycle when the PE is not blocked) are simulated to get estimates of effective throughput as well as latency.

Simulations have shown that buffers large enough to hold 6 messages give an effective throughput very close to the offered load, up to an offered load of about 0.3 messages per cycle for two-packet messages. Round-trip latency is less than that predicted; this seems to be due to smaller queueing delays on the return path. Combining keeps latency low even with heavy hot-spot traffic in a 64-PE network. However, we do not know the effect of clustered traffic to memory modules or ‘‘bursty’’ traffic.

In many practical cases, a PE may not issue another request until a previous request has been satisfied. The simulator can require a PE to wait for a previous load request to complete before making another. If one prefetch is allowed, a PE can have two load requests outstanding at the same time, and similarly for additional prefetching. Note that as the load increases, more prefetching greatly increases effective throughput (Figure 3).

3. Processing Element (PE) Design

This section presents a brief overview of the PE design for Ultra III, concentrating on interfacing an existing microprocessor to the network. For a more detailed discussion of other issues in PE design, see [8].

3.1 Processor Choice

A major issue in the design of the PE is the selection of microprocessor. Some of the relevant factors are as follows:

- PE design is impacted by the network which has high bandwidth but relatively high latency. To reduce the effect of the network’s latency a request for a data item should be issued as far

Offered Load	Effective throughput by amount of prefetch				Latency by amount of prefetch			
	None	One	Two	∞	None	One	Two	∞
0.1	0.05	0.08	0.09	0.09	15.8	16.2	16.4	16.5
0.3	0.07	0.13	0.17	0.23	16.2	17.2	18.3	20.4
0.5	0.08	0.15	0.20	0.33	16.3	17.7	19.2	27.5
0.7	0.08	0.15	0.21	0.41	16.4	17.9	19.7	43.4

Figure 3. Effective throughput (messages per cycle) and latency (cycles) for varying prefetch and offered load: 64 PEs, 2×2 switches, 20 packet queues, two-packet messages, and memory cycle time of two network cycles. Request mix: 60% loads, 40% stores. Unlimited stores allowed.

as possible in advance of its use. This is, in general, not easy and is especially difficult given a processor designed around small memory latency. As discussed above, even a small number of outstanding requests increase network utilization: the more outstanding requests allowed, the better the performance. [8] discusses both this issue and the design of the PE in more detail.

- Since no known microprocessor implements the required fetch-and-ϕ operations, the PE must interact with the processor to support them. However, the application provides a virtual address for the fetch-and-ϕ target, while the PE must transmit a physical address. If address translation is done inside the processor, the only efficient way to implement the operation is to issue a load operation specifying the target address of the fetch-and-ϕ and have the external hardware detect this bus cycle. This is not always simple and usually involves having the load be special in some way. In addition, caches (both on- and off-chip) must at least invalidate the fetch-and-ϕ target. The amount of ‘‘glue’’ logic required to implement the fetch-and-ϕ operation can be strongly dependent on otherwise insignificant details of processor architecture.
- Both a cache and floating-point processor are required on each PE. Preference is given to microprocessors that allow the construction of PEs with as few components as possible.

We have evaluated the microprocessors currently available against these criteria and chosen the AMD Am29000 Streamlined Instruction Processor. This is a RISC microprocessor performing at 17 MIPS when run at 25 MHz. Included in the family is the Am29027, a floating-point coprocessor that performs at 3.1 LINPACK MFLOPS (single-precision) and the Am29062, a sophisticated cache controller containing 8KB of data in 512 128-bit lines with a two-way set associative organization. (It is expected that both a larger cache and a processor with an on-chip floating point unit will be forthcoming.) The Am29062 can be programmed for various sub-block sizes, prefetch and write policies, and either write-back or write-through caching policies, in addition to many other options.

Some details of the design of the PE using the Am29000 series of chips is discussed in the remainder of this section; its structure is shown in Figure 4.

3.2 Using the Am29000 and Am29062

To perform fetch-and-ϕ operations, we take advantage of the fact that each Am29000 load and store instruction has an additional bit, the *lock* bit, that can be set by the user and has no internal effect, but is presented externally when the request is issued. We use this bit to flag fetch-and-ϕ operations. In such a case, the operation type and operand must be previously have been set by an I/O-space store; then the address is translated to a physical address by the normal operation of the load instruction. Presenting a transaction to the 29062 with the *lock* bit set causes it to

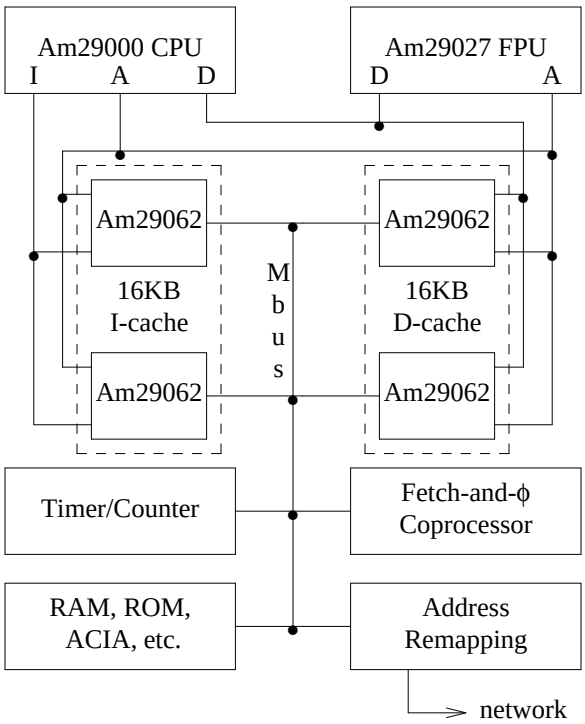

Figure 4. Structure of the Am29000 PE Board

invalidate the specified address if it is present; this is exactly the behavior we require for fetch-and-ϕ operations.

One of the key advantages of the Am29000 in systems such as Ultra III that have a PE-to-MM interconnection network with high bandwidth but high latency is that a PE using that processor (even without the 29062 cache) can have many load requests simultaneously outstanding in the network. One of the reasons for this is that the Am29000 has three 32-bit busses: address, instruction, and data. Addresses need not be presented during the entire transaction, so instruction and data references can proceed simultaneously. A ‘‘burst mode’’, where a single address is presented and subsequent accesses reference consecutive words, is also provided. In addition, the address of the next request can be placed on the address bus for previewing while a previous request is being processed; this is called a ‘‘pipelined’’ request. This ‘‘preview’’ can speed response time when the request is actually made. Up to three simultaneous load requests can be issued by the 29000: one primary request on each port (instruction and data) and a pipelined request on one of the ports. A network interface can prefetch additional words in advance of their being requested for burst requests. Finally, stores can often be acknowledged once they are transmitted to the network.

The 29000 has other features that minimize the effects of long memory latency for both instruction and data fetches. It prefetches instructions into a 4-word buffer, essentially pipelining instruction fetches, and, for data references, marks the destination register unavailable and allows execution to proceed until either the register is referenced or another load is issued. In the latter case, the second address is placed on the bus as a pipelined request.

An Am29000 simulator that tracks the prefetch and pipeline behavior confirms that we are likely to obtain significant processor utilization. Detailed analysis has not yet been done, but utilizations in the range of 50-75% seem likely. These translate into aggregate performance for a 64-PE Ultra III in the range of 500-800 MIPS if we can achieve a 25 MHz system clock speed.

In the Ultra III design, however, the network is not directly connected to the 29000, but rather to the caches, so a different class of requests are simultaneously outstanding. The 29062s are connected directly to the address, instruction, and data buses of the 29000 and are connected to a common ‘‘memory bus’’ which, in our case, is a network port. This bus multiplexes address and data lines and supports simultaneous read requests from multiple cache chips sharing the bus (write requests are acknowledged without waiting for the network response). Single-word load requests are issued as fetch-and-adds with zero increment and burst requests produce two double-word loads; the returned words are saved in a dual-ported RAM until requested by the 29062. Two load requests may be ouststanding from each of four Am29062s (two chips each for instruction and data), in addition to any pending stores and a ‘‘previewed’’ pipelined request, though it is doubtful whether any pattern of references from a single PE could exercise all possible load requests at the same time.

4. Memory Module (MM) Design

The logic of the MM is straightforward: it receives and services requests from the network[2]. 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. 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.

The major limitation on the MM cycle time is that of the memory chips. Restricting our attention to relatively inexpensive DRAMs yields MM cycle times in the range of 200-250ns. At 15MHz, the clock speed for our pre-prototype, this is within four clock cycles, 267ns. Since it takes four cycles to transmit a response to a 64-bit load (all messages must be an even number of packets), this is the minimum service time possible for that request. Improving the service rate to three cycles for the other requests is not possible since the network requires that messages be initiated on even numbered cycles and a service time of two cycles (133ns) seems unachievable with available memory chips. Simulations show four cycles to be adequate.

Since the MM is organized as a 1Mb×32 (plus parity) array, a 64-bit load requires two accesses. However, using ‘‘nibble-mode’’ to provide the second 32-bit word permits the DRAMs to perform these loads in four clock cycles. Fetch-and-ϕ operations requiring ALU operations cannot be completed in four cycles; these require six cycles. To permit loads to be combined with fetch-and-adds, the PEs transmit 32-bit loads as F&A’s of zero; this does not increase network traffic since the minimum message length is two packets. To perform 32-bit loads in four cycles, the MM checks for F&As with zero increments and suppress the write operation. Similarly, a 32-bit store is transmitted as a *fetch-and-store* and must also be performed in four cycles.

As the system clock speed increases towards the target clock rate of 25MHz, the service time of the memory, expressed in clock cycles, also increases. More detailed system modeling is planned, using results from early hardware to calibrate the simulations. Then it will be possible to determine whether the MM service rate of the present design provides adequate performance and if not whether faster RAMs or techniques such as caches or multiple memory banks on the MM should be employed.

[2]The MM also has a second port that connects to a back-end I/O processor. Because of the synchronous nature of the entire system, arbitration between the ports is not in the critical path and can be done in parallel with servicing requests.

5. Packaging

This section discusses the packaging of both the switch nodes and a 64-PE Ultra III with a combining network.

5.1 Switch Packaging

Each forward-path port requires 39 bits (37 for data and 2 for control); each return-path port 35. Thus, a single-chip 2×2 switch would require over 300 signal pins, an impractical number. With two chips per switching node, one for each direction, the maximum pin count is reduced. The forward path chip requires about 230 signal pins, the return path 210. These are within the capability of a single chip using modern packaging. Increasing the number of chips per 2×2 switching node to six requires even fewer pins (175 for the return path parts).

The final switch packaging will depend on the technology available to us. We are exploring the possibility of using either ceramic or silicon-substrate packaging to construct a 4×4 subnetwork which would need 600 signal pins, within the range of such packaging.

5.2 System Packaging

A simple modular packaging scheme for Ultracomputers has been developed using Cinch's ''Button'' technology [17] as used by MCC's ''ES-kit'' program [12]. This technology facilitates the interconnection of stacked modules with separate sets of connections on the top and bottom faces of each module. By appropriate partitioning of the network, a simple set of ''Ultra'' modules can be used to assemble Ultracomputers of different sizes.

A Button contact is fabricated from strands of 2 mil diameter silver-copper wire randomly pressed to form a cylinder 50 mils in diameter and 60 mils high. One Button is attached to each end of a conducting rod about an inch long. These rods contact 50 mil diameter gold-plated ''lands'' on adjacent PC-boards. Only pressure is required to complete the circuit (approximately one to two ounces per button). The assembly has a contact resistance of less than 0.3mΩ and is electrically invisible at frequencies up to that of microwaves. It can be used to distribute either power or signals.

A standard connector developed jointly by Cinch and MCC provides four rows of Buttons on 100 mil centers with a 50 mil offset between rows. This connector consists of 106 rods described as above and has a ''footprint'' of 3.5″×.5″, including alignment pins and bolt holes. With these connectors, standard PC-board fabrication technology is used to produce an appropriate pattern of 50-mil lands, and holes are drilled for alignment pins and compression bolts.

There are many ways to construct Ultracomputers using this technology. They can be evaluated by comparing the total number of boards required, the total number of board *types* required, the total number of Buttons required, the size of the largest board, or other similar criteria.

An Ω network of N ports has the property that, for any power of two, $i \leq \sqrt{N}$, a block containing an $i \times i$ network and either i PEs or i MMs can be constructed where all the switch connections are internal to the block and i ports exit from the block. In the case of a 64-PE Ultra III, we can construct basic blocks of four PEs or MMs that include a 4×4 network.

Three basic module types are necessary to form a four processor Ultracomputer: a processor (PE) module, a memory (MM) module, and a four-input, four-output switch (SW) module. As shown in Figure 5, each 7" side contains a pair of Button connectors on each board surface. Since each network port (PE-to-MM plus MM-to-PE paths) requires 74 wires and each connector provides 106 contacts, each connector is reserved for a single port. Each board can therefore have four ports per surface and we can construct a board containing four 2×2 switches connected to

Figure 5. Top or bottom view of basic Ultra III board. Top connectors need not be connected to the corresponding bottom connectors. 106 contacts per connector.

form a 4×4 sub-network with its ‘‘PE’’ ports on the top surface and its ‘‘MM’’ ports on the bottom.

Both the PE and MM modules are wired so that the external port is located in one corner of the top face and also wired to a corner of the bottom face rotated by 90 degrees. The remaining three top connections are separately wired from a top corner to a 90 degree rotated bottom corner (Figure 6). The ports to the network from four stacked PEs or MMs are thus routed through the intermediate boards to both the top and bottom faces of the stack.

The four-input, four-output switch (SW) module has its inputs wired to each of the four top corners and its outputs to the corresponding four bottom corners (Figure 7). A stack of nine modules (4 PEs, 1 SW and 4 MMs) comprises a 4 PE Ultracomputer, a useful assembly for diagnosis of PE/MM operation.

Larger Ultracomputers can be assembled using simple wire routing boards. Two types of routers are required to connect 4×4 switch stages, one type vertical and the other horizontal. By connecting them orthogonally, long board-to-board wires are avoided (see [1], [19]). Taking the edge length of the other modules as a unit, these wire routers have a small dimension of one unit and a large dimension of two, four or eight units depending on the number of stages required.

For example, to build a 16 PE Ultracomputer, only two kinds of routers are needed, a horizontal router (hR), one unit high by two units wide, and a vertical router (vR) two units high by one unit wide (Figure 8). Two of each are required. These routers distribute each of the four outputs of a 4×4 switch stage to one input of each of four different 4×4 switches of the next stage. A 16 PE Ultracomputer has four stacks, each with 4 PEs and one SW; a layer of two hRs; a layer of two vRs; and four stacks, each having one SW and 4 MMs (Figure 9).

In a 64 PE Ultracomputer (Figure 10), larger HR and VR routers (4×1 units) and a stack height of 13 boards (4 PEs, SW, VR, SW, HR, SW, 4 MMs) are required. Assuming one inch spacing between board centers, each stack is 13 inches high. Not including power supplies and

Figure 6. MM/PE Board Figure 7. 4×4 Switch

Figure 8. Horizontal and Vertical Routers

Figure 9. 16-PE Ultracomputer

Figure 10. 64-PE Ultracomputer

fans, a 64 PE Ultracomputer takes up 40 inches by 28 inches by 13 inches, less than one third cubic yard. Estimated power consumption is about 1500 watts, well within the limits of forced-air cooling using standard fans.

This method of system interconnection can be extended to 256 PEs. Such a system would have 16 boards in a stack and dimensions 80 inches by 56 inches by 16 inches. For still larger systems, a basic stack of 8 or 16 PEs and a corresponding switch is desirable to minimize the maximum wire length.

References

[1] R. Bianchini and R. Bianchini, Jr., ‘‘Wireability of an Ultracomputer,’’ Research and Development Report, Courant Mathematics and Computing Library, New York University, October 1982.

[2] A. Borodin and J. E. Hopcroft, ‘‘Routing, Merging, and Sorting on Parallel Models of Computation’’, *Proc. of the 14th Annual ACM Symp. on Theory of Comp.*, pp. 338-344, 1982.

[3] S. Dickey, R. Kenner, M. Snir, and J. Solworth, ‘‘A VLSI Combining Network for the NYU Ultracomputer’’, *Proc. of the IEEE Intl. Conf. on Comp. Design*, pp. 110-113, 1985.

[4] S. Dickey and Y. Liu, ‘‘Simulation and Analysis of Different Switch Architectures for Interconnection Networks in MIMD Shared Memory Machines’’, NYU Ultracomputer Note No. 141, June 1988.

[5] L. R. Goke and G. J. Lipovski, ‘‘Banyan Networks for Partitioning Multiprocessor Systems’’, *Proc. of the First Ann. Symp. on Comp. Arch.*, pp. 21-28, 1973.

[6] A. Gottlieb, ‘‘An Overview of the NYU Ultracomputer Project’’, *Experimental Parallel Computing Architectures*, J. Dongarra, ed., North Holland, pp. 25-95, 1987.

[7] A. Gottlieb, B. Lubachevsky, and L. Rudolph, ‘‘Basic Techniques for the Efficient Coordination of Very Large Numbers of Cooperating Sequential Processors’’, *ACM TOPLAS*, Vol. 5, No. 2, pp. 164-189, Apr. 1983.

[8] R. Kenner, S. Dickey and P. Teller, ‘‘The Design of Processing Elements on a Multiprocessor System with a High-bandwidth, High-Latency Interconnection Network,’’ *Proceedings of the Hawaii International Conference on System Sciences*, January, 1989.

[9] C. Kruskal and M. Snir, ‘‘The Performance of Multistage Interconnection Networks for Multiprocessors’’, *IEEE Trans. on Computing*, Vol. C-32, No. 12, pp. 1091-1098, Dec. 1983.

[10] C. Kruskal, M. Snir, and A. Weiss, ‘‘The Distribution of Waiting Time in Clocked Multistage Interconnection Networks’’, *Proc. of the 1986 Intl. Conf. on Parallel Processing*, pp. 12-19.

[11] D. Lawrie, ‘‘Access and Alignment of Data in an Array Processor’’, *IEEE Trans. on Comp.* Vol. C-24, No. 12, pp. 1145-1155, Dec. 1975.

[12] MCC Technical Report Number: ACA-ESP-391-87, ‘‘ES-Kit Project Overview’’, December 21, 1987.

[13] A. Norton and G. Pfister, ‘‘A Methodology for Predicting Multiprocessor Performance ’’, *Proc. of the 1985 Intl. Conf. on Parallel Processing*, pp. 772-781.

[14] O. Percus and J. Percus, ‘‘Elementary properties of Clock-regulated Queues,’’ NYU Ultracomputer Note #138, May 1988.

[15] G. Pfister and A. Norton, ‘‘‘Hot Spot’ Contention and Combining in Multistage Interconnect Networks’’, *IEEE Trans. on Comp.*, Vol. C-34, No. 10, pp. 943-948, Oct. 1985.

[16] J. T. Schwartz, ‘‘Ultracomputers’’, *ACM TOPLAS*, Vol. 2, pp. 484-521, 1980.

[17] R. Smolley, *Button Board: A New Technology Interconnect for 2 and 3 Dimensional Packaging*, TRW Technical Report, presented at Int. Soc. for Hybrid Microelectronics, Nov. 1985.

[18] J. Wilson, *Operating System Data Structures for Shared-Memory MIMD Machines with Fetch-and-Add*, Ph.D. Thesis, Courant Institute, NYU, June 1988.

[19] D. S. Wise, ‘‘Compact Layout of Banyan/FFT Networks’’, *CMU Conference on VLSI Systems and Computations*, Kung, Sproull and Steeler (eds.), Computer Science Press, Rockville, Maryland, 1981, pp. 186-195.