[16] Stephen K. Park and Keith W. Miller. Random number generators: Good ones are hard to find. *Communications of the ACM*, 31(10):1192 – 1201, October 1988.

[17] Howard J. Siegel, Wayne G. Nation, Clyde P. Kruskal, and Leonard M. Napolitano. Using the multistage cube network topology in parallel supercomputers. In *Interconnection Networks for Large-Scale Parallel Processing* by Howard J. Siegel, McGraw-Hill Publishing Company, New York, 1990. Pages 313 – 364.

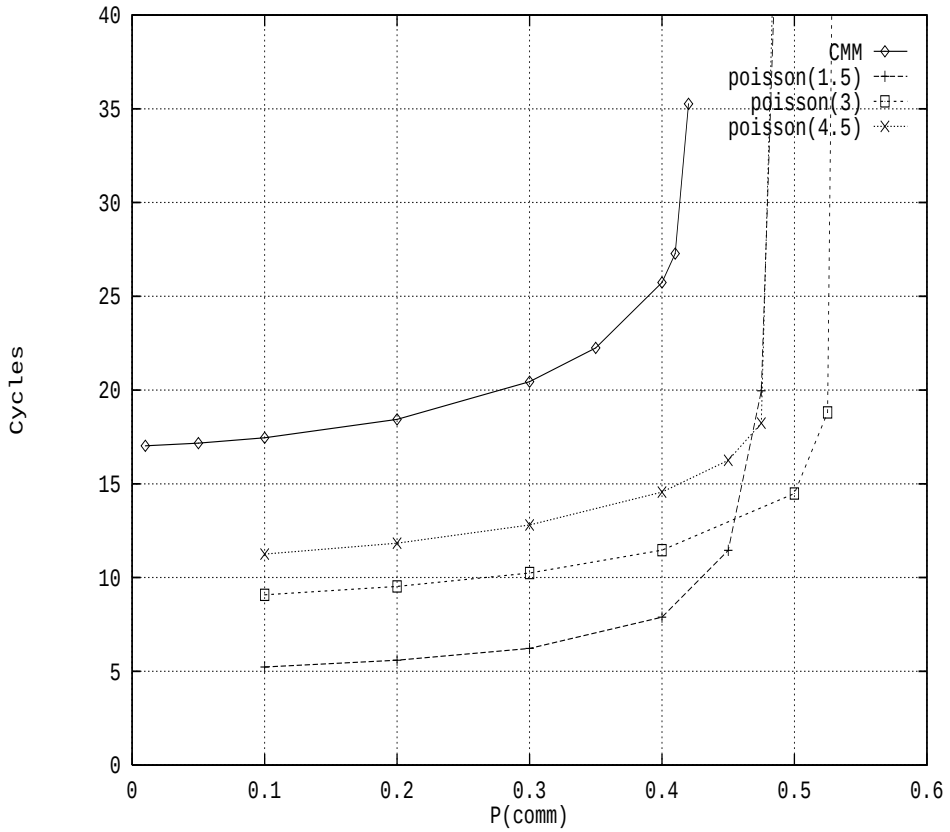

Figure 12: Poisson distributions, 128 PEs

[10] D. N. Jayasimha and J. D. Martens. Some architectural and compilation issues in the design of hierarchical shared memory multiprocessors. In *International Parallel Processing Symposium*, pages 567 – 572, March 1992.

[11] Averill M. Law and W. David Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, New York, 1982.

[12] Gyungho Lee, Clyde P. Kruskal, and David J. Kuck. The effectiveness of combining in shared memory parallel computers in the presence of "hot spots". In *International Conference on Parallel Processing*, pages 35 – 41, 1986.

[13] Stephen A. Mabbs and Kevin E. Forward. Optimizing the communication architecture of a hierarchical parallel processor. In *International Conference on Parallel Processing*, volume I, pages 516 – 520, August 1990.

[14] J. D. Martens and D. N. Jayasimha. A tree-structured hierarchical memory multiprocessor. Technical Report 61, Department of Computer and Information Science, The Ohio State University, December 1989. Revised May 1990. An extended abstract appeared in *International Conference on Parallel Processing*, volume I, pages 561 – 562, August 1990.

[15] Haim E. Mizrahi, Jean-Loop Baer, Edward D. Lazowska, and John Zahorjan. Introducing memory into the switch elements of multiprocessor interconnection networks. *International Symposium on Computer Architecture*, pages 158 – 166, 1989.

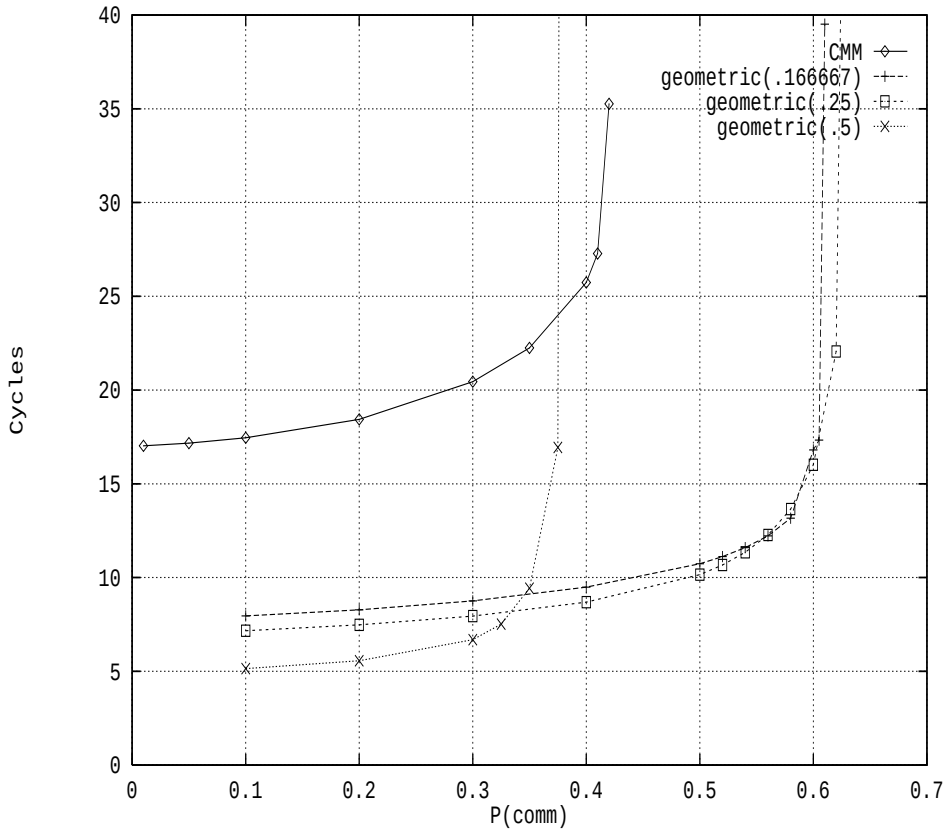

Figure 11: Geometric distributions, 128 PEs

[2] G. S. Almasi and A. Gottlieb. *Highly Parallel Computing.* Benjamin/Cummings, New York, 1989.

[3] Arvind and Robert A. Iannucci. Two fundamental issues in multiprocessing. Technical Report 226-6, Laboratory for Computer Science, MIT, May 1987.

[4] Jianxun Ding and Laxmi N. Bhuyan. Performance evaluation of multistage interconnection networks with finite buffers. In *International Conference on Parallel Processing*, volume I, pages 592 – 599, August 1991.

[5] Tse-yun Feng. A survey of interconnection networks. *IEEE Computer*, pages 12 – 27, December 1981.

[6] Harlan E. Husmann. *Compiler Memory Management and Compound Function Definition for Multiprocessors.* PhD thesis, University of Illinois, Urbana, August 1986.

[7] Raj Jain. *The Art of Computer Systems Performance Analysis.* Wiley, New York, 1991.

[8] D. N. Jayasimha. Distributed synchronizers. In *International Conference on Parallel Processing*, pages 23 – 27, August 1988.

[9] D. N. Jayasimha. Partially shared variables and hierarchical shared memory multiprocessor architectures. In *11th Annual IEEE International Conference on Computers and Communications*, pages 63 – 71, April 1992.

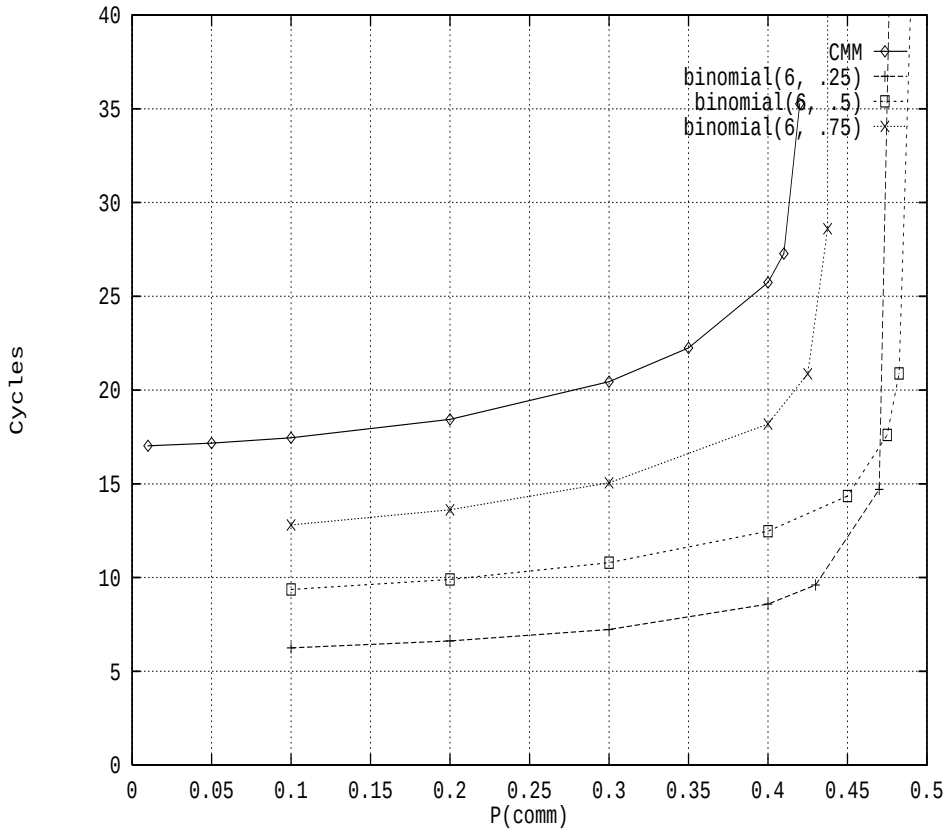

Figure 10: Binomial distributions, 128 PEs

believe that hierarchical architectures such as the THMM will be useful for the reduction of latency and synchronization delays in large-scale shared memory multiprocessors, thus helping to overcome the so called "von Neumann bottleneck." We saw that the THMM performed particularly well compared to the CMM for some triangular distributions; we expect variables to be distributed in this manner with the lower and middle levels of the hierarchy being used more than the higher levels. A number of research issues remain to be investigated, however. For instance, the current simulation restricted itself to a synthetic workload. As with any stochastic simulation, there is some question whether any particular workload is truly representative of any actual program. Thus, modifying the simulator to execute specific programs for both the THMM and the CMM is a near-term goal. With the current synthetic workload, it may be interesting to investigate placing two memory modules at each switch as mentioned in section 3.5.3, and to further investigate the effect of limited overtaking mentioned in section 3.2.1. Additionally, it may be worthwhile to study the use of $k \times k$ switches in the THMM for values of k other than 2.

Currently, we are looking at strategies for compiling nested iterative loops on the THMM, studying the performance of individual algorithms on the THMM, and studying the mapping of algorithms to the THMM.

References

[1] Alok Aggarwal, Bowen Alpern, Ashok K. Chandra, and Marc Snir. A model for hierarchical memory. In *19th Annual ACM Symposium on Theory of Computing*, 1987.

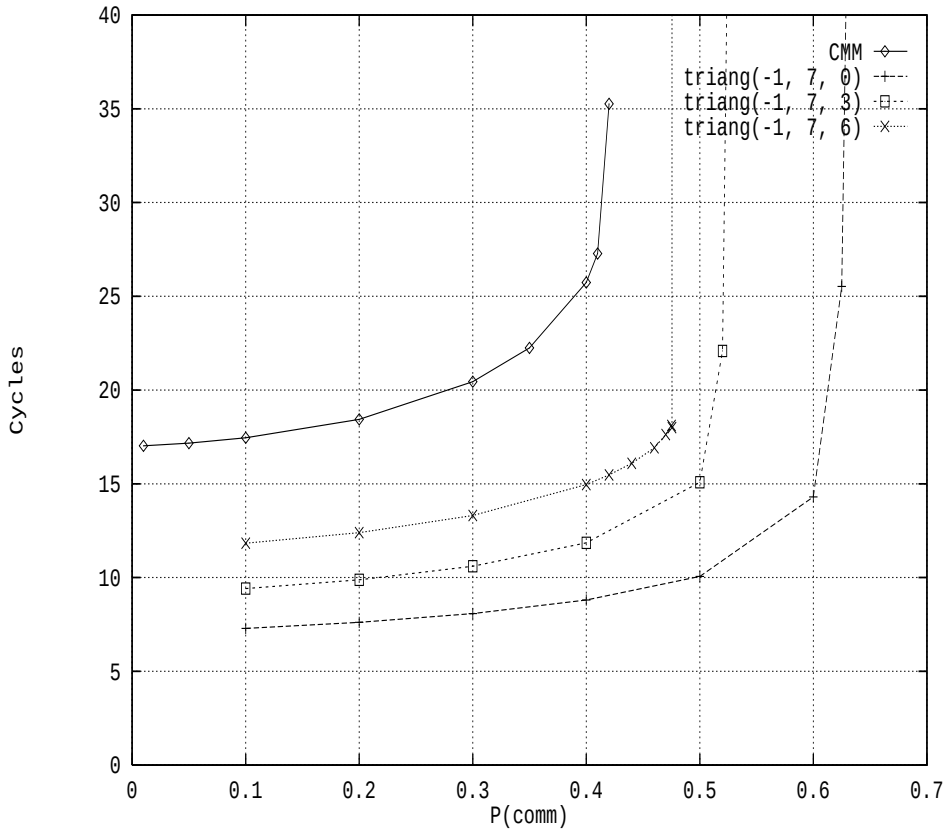

Figure 9: Triangular distributions, 128 PEs

Figure 11 shows the performance of the THMM relative to the CMM under the assumption that the level number of each request is drawn from a truncated $geom(p)$ distribution [11] for $p \in \{\frac{1}{6}, \frac{1}{4}, \frac{1}{2}\}$. Although the THMM outperforms the CMM for $geom(\frac{1}{6})$ and for $geom(\frac{1}{4})$, it saturates at a lower communication rate than the CMM for the $geom(\frac{1}{2})$ distribution. This is due to the fact that the vast majority of the memory requests are very low in the memory hierarchy, and each memory becomes a single server for two queues, producing a bottleneck. This could be fixed by placing two memories at each switch—effectively having as many servers as queues.

3.5.4 Poisson Distribution

Figure 12 shows the performance of the THMM relative to the CMM under the assumption that the level number of the requests is drawn from a truncated $Poisson(\lambda)$ distribution [11] for $\lambda \in \{1.5, 3, 4.5\}$, which places the mean distance of access at one-fourth, one-half, and three-fourths of the switch stages, respectively. For the $Poisson(1.5)$ distribution, we can again see the effect of the memory units closer to the PEs becoming a bottleneck.

4 Conclusion and Future Research

Based on analytical results presented elsewhere [9] and on the simulations discussed above, we

mance of the THMM relative to the CMM should be greater than that suggested in the preceding sections. With longer dependence distances, however, the average communication distance may not be this small. Furthermore, if a loop is not amenable to static analysis, then the compiler cannot determine which PEs will access a particular data item. In this case, a conservative placement of the datum may cause the average distance of reference to increase.

Other algorithms which were investigated in [10] include matrix multiplication and parallel prefix. A scheme is presented for matrix multiplication that accesses all levels of the hierarchy evenly. The parallel prefix algorithm accesses higher levels of the hierarchy more than lower levels.

Since the access level may not be uniformly distributed, we study the performance of the THMM with the generation of requests to different levels of the hierarchy drawn from a variety of distributions below. These distributions are binomial, geometric, Poisson, and discrete triangular.[6]

The aforementioned distributions were chosen because they are commonly used discrete distributions. Furthermore, certain of these distributions are intuitively similar to the expected behavior of a particular class of program on the THMM. In general, three sets of simulations were run for each distribution, with the parameters to the distributions chosen to place the mean access level low in the hierarchy (close to the PEs), near the middle of the hierarchy, and near the global level of the hierarchy.

In each of the following simulations, we assume that both the CMM and THMM consist of 128 PEs. The memory unit at a given level is chosen uniformly; communications are also generated uniformly. The level of an access is chosen from the specified nonuniform distribution. P(blocking) is zero in each case.

3.5.1 Triangular Distribution

Figure 9 shows the performance of the THMM relative to the CMM under the assumption that the level number of the requests is drawn from a discrete $triang(a, b, c)$ distribution [11], where $a = -1$ is the lower extreme for the range, $b = 7$ is the upper extreme of the range, and $c \in \{0, 3, 6\}$ is the mode. The THMM performs well under these distributions. We expect (as a result of looking at a number of algorithms) that the distribution of variables in the hierarchy will follow triangular distributions with modal values skewed toward the lower levels of the memory (the two bottommost plots in figure 9).

3.5.2 Binomial Distribution

Figure 10 shows the performance of a 128 PE THMM as compared to the CMM under the assumption that the level number of each request is drawn from a $bin(t, p)$ distribution [7, 11] for $t = 6$ (for switch stages zero through six) and $p \in \{\frac{1}{4}, \frac{1}{2}, \frac{3}{4}\}$. The mean distance of access (tp) for these three distributions is one-fourth, one-half, and three-fourths of the switch stages, respectively. Under any of these distributions, the THMM clearly outperforms the CMM.

3.5.3 Geometric Distribution

[6] Standard Poisson and geometric distributions are in the range $\{0, 1, 2, \ldots\}$, whereas we have only a finite number of levels in our hierarchy. Because of this, these distributions have been generated from a truncated distribution using the inverse transform method, as suggested by Law and Kelton [11].

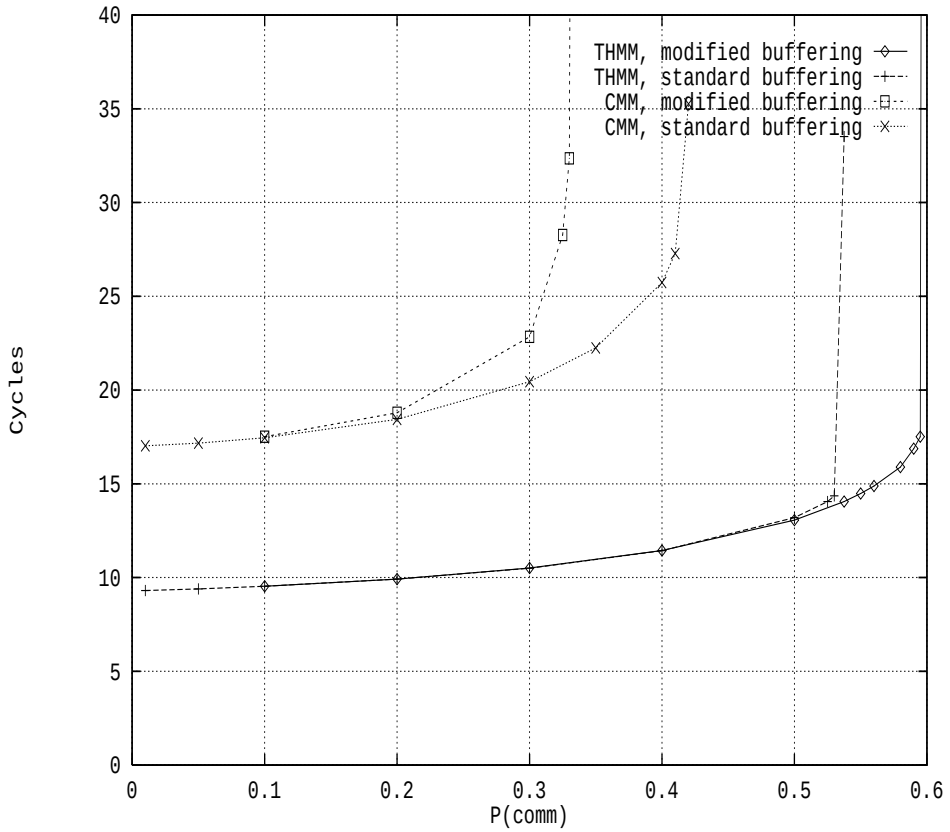

Figure 8: Service time vs. P(comm), $p = 128$, P(blocking) = 0, large buffers closer to PEs

program being executed. For example, consider a parallel sort algorithm [14]. Each PE can be given a sublist of the items to be sorted, and after each PE sorts its sublist, the sublists are merged in parallel. Each of the $\log p$ merge steps takes place at a successively higher level of the hierarchy, and each involves a pair of longer sublists than in the preceding step. Thus, at each step of the merge more accesses occur per pair of sublists merged, and the expense of each access is greater. On the other hand, at each successive step fewer merges are being performed in parallel. So the total number of accesses at each level is the same.

Another important algorithm is five point differencing. A static analysis of this performance on the THMM vs. the CMM was given in [14]. Here, the distance of the average nonlocal communication is significantly less than $\frac{1}{2}\log p$. Thus the THMM's performance relative to the CMM should be better than that obtained in the simulation model.

Vector reduction was also considered in [14]. Some PE must access each level of the hierarchy approximately the same small constant number of times, but not every PE must access every level. Traffic will again be concentrated in the lower levels of the hierarchy, but since the average access distance of the PE which performs the most work (the one that actually calculates the final sum) the behavior of this algorithm or similar algorithms should resemble that of our simulation model.

For doacross loops with a dependence distance of one every iteration must synchronize with adjacent iterations. In this case, if we statically assign adjacent iterations iterations to successively numbered processors, the assumption that levels in the hierarchy are chosen uniformly is pessimistic: half the synchronizations are in the lowest level of the hierarchy, half of the remaining synchronizations can take place in the next level of the hierarchy, and so forth. Thus, the perfor-

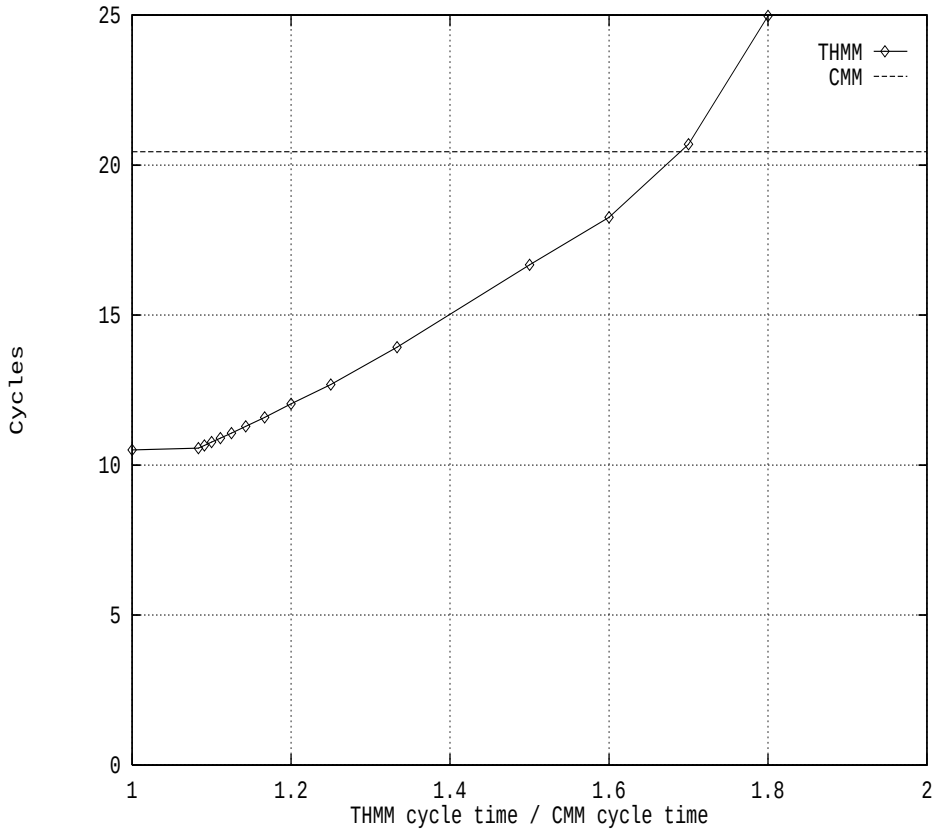

Figure 7: Service time vs. THMM cycle time, $p = 128$, P(blocking) = 0, P(comm) = 0.3.

by providing more buffering near the PEs and less near the global level. A similar modification should degrade CMM performance, however, since the traffic within the CMM's interconnect is independent of switch stage.

In figure 8 we see the effect of making the buffer size approximately proportional to the traffic within the THMM for $p = 128$, P(blocking) = 0, and various values of P(comm). For the seven stage network, this would have queue sizes of (7, 6, 5, 4, 3, 2, 1), keeping the total buffer space constant. For implementation reasons, we instead simulated queue sizes of (6, 6, 5, 4, 3, 2, 2). As can be seen in the figure, the THMM performs slightly better with this buffering scheme under high traffic loads, saturating at P(comm) ≈ 0.59 rather than P(comm) ≈ 0.54. The two curves separate as the buffers in switch stages close to the PEs fill; when the buffers are nearly empty, there is no difference between the two schemes. As expected, the CMM's performance is seriously degraded by performing a similar modification.

3.5 Differing Distributions of Accesses to Various Levels

Until now it has been assumed that the addresses of requests are uniformly distributed. Among memory units at a given level (or the global memory of the CMM), this is reasonable since the programmer or compiler will attempt to spread data evenly among memory units in an attempt to minimize contention. The justification for the assumption of uniform request rates among levels of the THMM is less straightforward. The validity of this assumption depends entirely on the

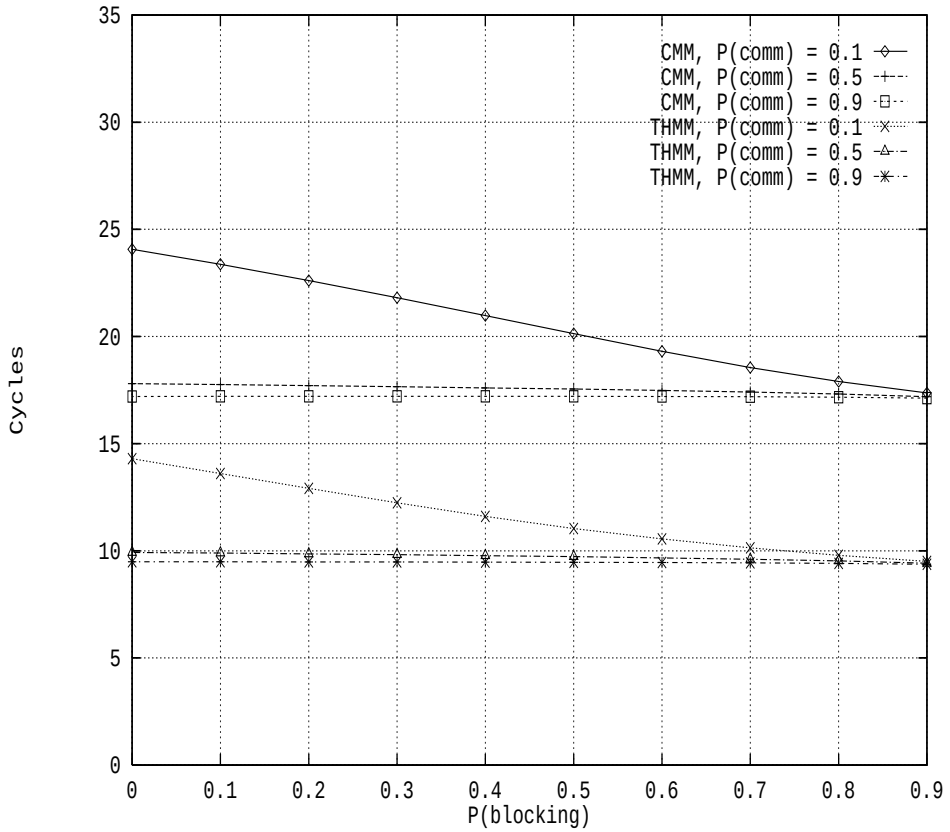

Figure 6: Service time vs. P(blocking) for $p = 128$, various values of P(comm).

of determining whether to forward a packet to the next stage. In figure 7, we see the effect of slowing the THMM network with respect to that of the CMM. The horizontal line at delay ≈ 20.4 cycles is the CMM with relative network cycle time of one. At this traffic load (P(blocking) = 0, P(comm) = 0.3), it appears that a 128 PE THMM will outperform a 128 PE CMM as long as the cycle time of the THMM is within a factor of 1.7 of the CMM. In actuality, we expect the THMM's network cycle time to be within 10 to 15 percent of the CMM's.

The break-even point of 1.7 is slightly lower than the ratio between the performance of the THMM to the CMM for 128 PEs given in figure 4. This is due to the fact that PEs continue to generate requests at the old rate even with a slower network, and thus the network is more prone to congestion as its cycle time increases relative to the processor cycle time.

3.4 Effects of Modified Buffering

Under the uniform traffic model, the stages of the THMM's interconnect closest to the PEs carry a majority of the traffic. Indeed, every packet must pass through the stage closest to the PEs, all but $\frac{1}{\log p}$ of the packets must pass through the next stage, and so forth. If N is the total number of packets generated over a simulation run, and d is the distance in network hops of a stage from the PEs, then the expected number of packets passing through the stage is $N\frac{\log p - d + 1}{\log p}$. An effect of this that is that the average queue lengths in switches close to the PEs are large, but queue lengths far from the PEs are small. This suggests that we can make better use of resources within the THMM

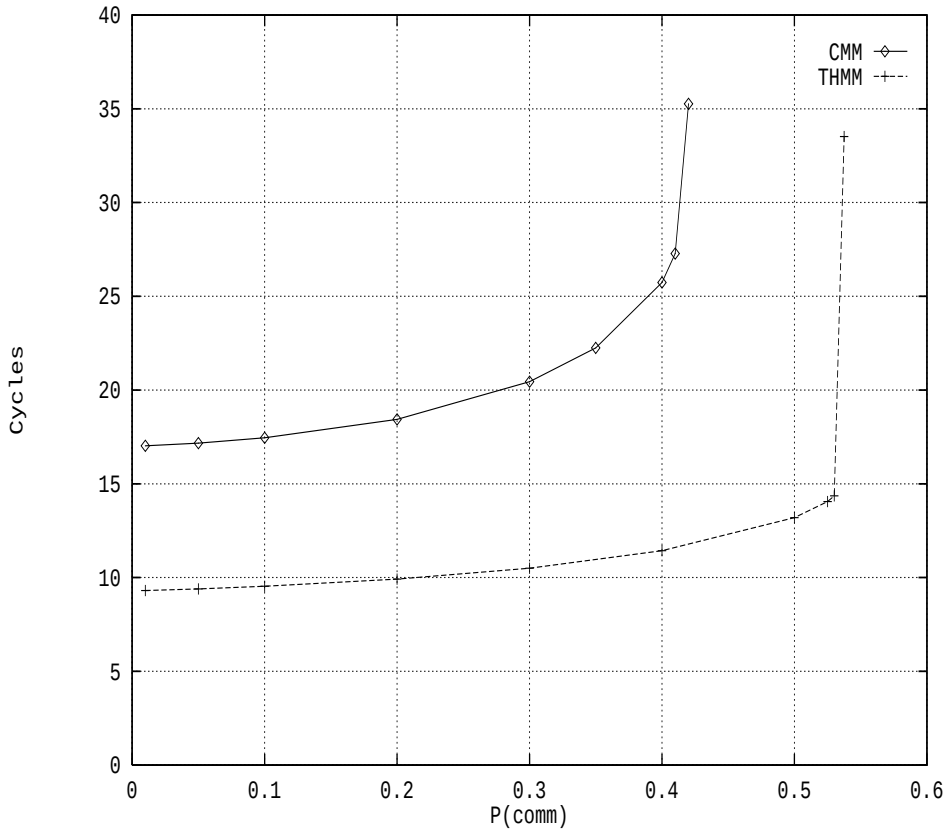

Figure 5: Service time vs. P(comm) for $p = 128$.

for the solution of those communication-intensive problems that can be effectively mapped to the hierarchy.

3.2.3 Blocking versus Nonblocking Requests

On real multiprocessors, communication is typically not generated at a steady rate. For example, in some cases, after generating a packet the processor will wait for a response before continuing; in other cases, the processor will continue without pause after a packet is sent. In figure 6 we see delay as a function of P(comm) for probabilities of blocking communication of 0.1, 0.5, and 0.9 for a CMM and a THMM with 128 PEs each. As can be seen from the figure, contention plays a minor role at what are effectively quite low communication rates. Even though P(comm) = 1, contention has little impact with P(blocking) ≥ 0.5. This is because although the PE generates a communication with probability P(comm) whenever it is not waiting for a blocking communication, each PE spends spends much of its time waiting. For low P(blocking), contention is more of a factor in both architectures.

3.3 Effects of a Slow THMM Network

It is possible that, in an implementation of the THMM, the network cycle time will be longer than on a similarly sized CMM. A potential source of this longer cycle time is the added complexity

but blocked by packets waiting for entry to a full switch memory queue are forwarded. With buffer sizes of four, this limited overtaking had little effect.

Figure 4 shows the ratio of the CMM delay to the THMM delay for the points plotted in figure 3.[5] Neglecting contention, we would expect this to be just under two (approximately $\frac{1+2\log p}{1+\log p}$), since the level in the THMM hierarchy is chosen uniformly. For the low-contention cases, P(comm) = 0.2 and P(comm) = 0.3, this is approximately correct. For P(comm) = 0.4, the relative performance of the THMM is better than would be expected if contention were neglected; this is because at any stage of the THMM's forward network but the first, there are fewer total requests, and thus less contention.

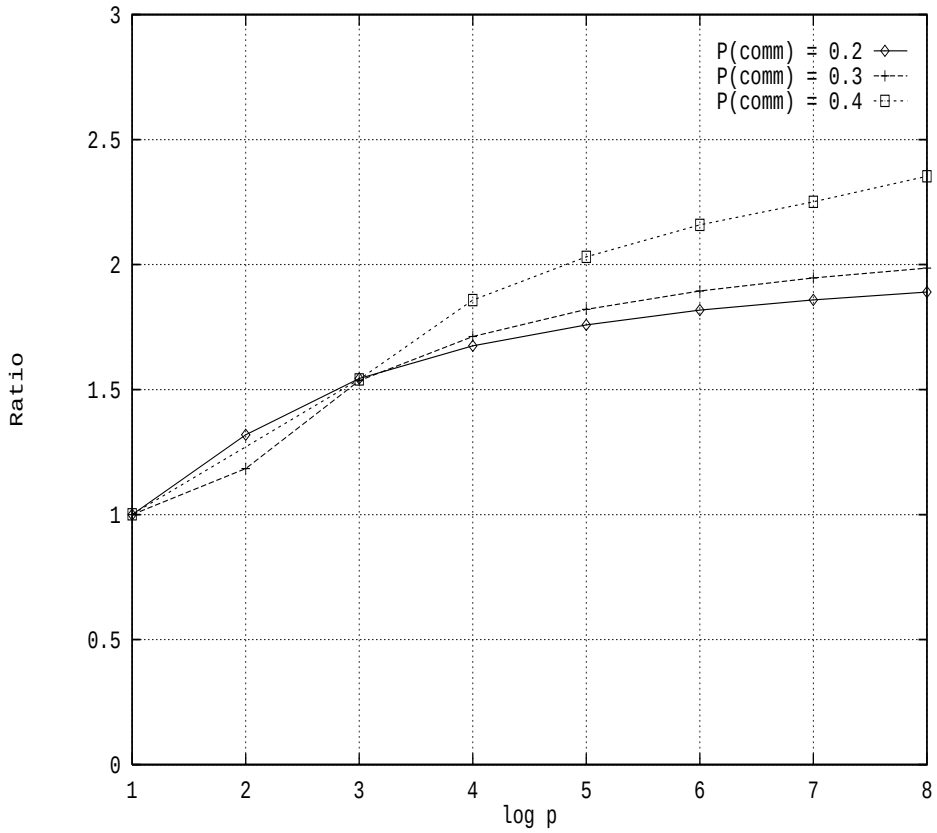

Figure 4: Ratio of service times vs. $\log p$ for P(comm) $\in \{0.2, 0.3, 0.4\}$.

3.2.2 Delay versus Network Traffic

Figure 5 shows delay as a function of P(comm) for 128 PEs. Notice that as network traffic increases, the gap between the THMM and the CMM also increases; this is consistent with the data presented in figure 4, and is due to shorter average queue lengths in the THMM's forward network. The CMM saturates at a request rate of approximately 0.42, the THMM at just less than 0.54. This suggests that a THMM of this size will be less adversely impacted by high traffic or bursts of traffic on the network than a CMM of the same size, making the THMM more suitable

[5] In order to maintain a small scale on the graph, we again do not plot the $p = 4$, P(comm) = 0.4 case.

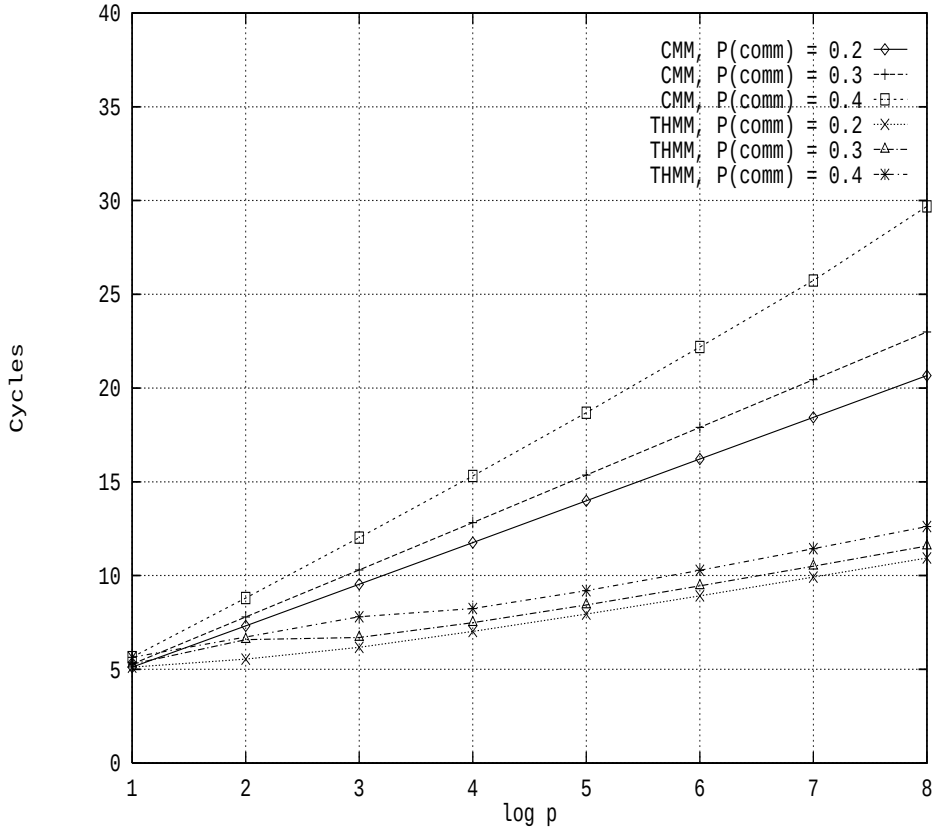

Figure 3: Service time vs. $\log p$ for P(comm) $\in \{0.2, 0.3, 0.4\}$.

THMM switch-memory element (figure 2) and looking at queue lengths for this simulation (table II). The poor performance of the THMM in this case is due to long queue lengths for the THMM

	THMM	CMM
Avg. Switch Input Queue Length, Forward Stage 0	2.60	0.408
Avg. Switch Input Queue Length, Forward Stage 1	0.0981	0.415
Avg. Global Memory Input Queue Length	0.000072	0.00128
Avg. Switch Input Queue Length, Return Stage 0	0.0980	0.376
Avg. Switch Input Queue Length, Return Stage 1	0.214	0.303
Avg. PE Input Queue Length	0.389	0.400

Table II: Queue lengths for the $p = 4$, P(comm) $= 0.4$ case.

in the forward network, caused by the memory queue in the switch becoming full and thus refusing packets. Whenever the memory queue in the THMM switch refuses a packet, the blocked packet remains at the head of the switch's input queue. Since our queues are strictly FIFO, this blocks that input to the switch. When p is just four, a large portion of the forward network is blocked whenever this happens. For P(comm) of 0.4 or less and higher processor counts, this behavior is not seen—presumably because it is a transient behavior involving only a small fraction of the network. As a possible solution to this problem, a limited form of overtaking in the queues was implemented; we term this limited overtaking, since only packets destined for a free switch output

in figure 3.

p	P(comm) = 0.2		P(comm) = 0.3		P(comm) = 0.4	
	CMM	THMM	CMM	THMM	CMM	THMM
2	5.11	5.12	5.28	5.28	5.65	5.64
4	7.32	5.55	7.79	6.58	8.81	2660
8	9.54	6.18	10.3	6.70	12.0	7.80
16	11.8	7.02	12.8	7.49	15.3	8.24
32	14.0	7.95	15.4	8.44	18.7	9.20
64	16.2	8.92	17.9	9.45	22.2	10.3
128	18.4	9.92	20.4	10.5	25.7	11.4
256	20.7	10.9	23.0	11.6	29.7	12.6

Table I: Delay as a function of p for various communication rates.

In the paragraphs below, we first examine delay as a function of number of PEs. This is done in order to compare the scalability of the two architectures. For these simulations, p varies from 2 to 256, and three values of P(comm) are investigated.

We then turn our attention to delay as a function of network traffic. As the traffic increases, the amount of contention will increase, and the delay should likewise increase. At some point, the communication will overwhelm the network, and the network will become saturated. This will take the form of an asymptote in the graph past which delay is unbounded. For example, Siegel, et al. analytically predict a throughput of approximately 0.3 packets per processor per cycle on an unbuffered network with 128 processors and 2×2 switch switch elements [17]; this corresponds to P(comm) = 0.3 and P(blocking) = 0 in our model. As can be seen below, the CMM does somewhat better than this, presumably due to the input buffers in the CMM's switches. Our study of delay vs. network traffic is performed with $p = 128$, P(blocking) = 0, and various values of P(comm).

We then consider the fact that many communications are blocking, i.e., that often a computation cannot proceed until the result of a read is known or until a write acknowledgment is received. Thus, we vary P(blocking) while holding other simulation parameters constant.

This is followed by an investigation of how CMM performance compares to THMM performance if the THMM's interconnection network is slower than the CMM's, and we then consider a modified buffering scheme for the THMM.

Finally, we discard the assumption that the distance into the hierarchy that a communication must travel is uniformly distributed. The level of a THMM's access is drawn from various distributions, and the resulting delay is compared to that of the CMM.

3.2.1 Delay versus Processor Count

Table I and figure 3 show delay as function of the number of switch stages, $\log p$ with three values of P(comm) for both the THMM and the CMM.

Under almost all circumstances, the round-trip (or service) time of a packet from a PE to a memory and back to the PE is considerably lower on the THMM than for the corresponding CMM case. The exception to this is the $p = 4$ and P(comm) = 0.4 case, where delay is lower for the CMM than the THMM.[4] The reason for this can be seen by examining the structure of the

[4]Indeed, the THMM simulation is unstable under these circumstances, and the delay increases with longer simulation runs. Since the delay measured for the $p = 4$ and P(comm) = 0.4 case is not of the same order of magnitude as the other delays, and since we are primarily interested in large-scale multiprocessors, this point has not been included

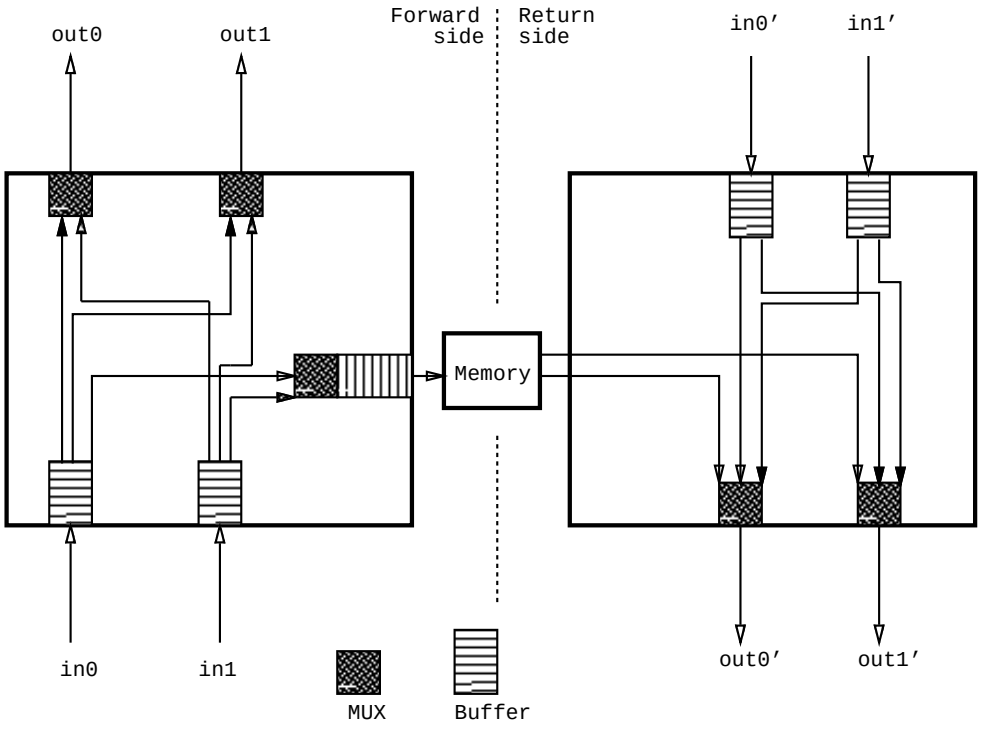

Figure 2: The THMM forward and return networks connect through a switch-memory element.

overlapped, low average latency will reduce the number of instructions which must be overlapped with each communication in order to obtain good performance.

Parameters to the various simulation runs include:

- p: the size of the multiprocessor, measured in number of PEs.
- P(comm): the probability that a given processor will generate a communication in any single cycle, i.e., generate a non-local memory request.
- P(blocking): the probability that a given request will be blocking, i.e., that the processor must wait for the request to be satisfied before generating additional non-local memory requests.

Note that if P(blocking) is zero, then P(comm) is the actual request rate—if the network is unable to accept a packet from a PE, the packet is queued for insertion into the network as soon as there is space, and the PE continues as if the packet had been transmitted.[3] If P(blocking) is nonzero, then P(comm) represents the conditional probability that a given processor will generate a request given that it is not already waiting for an acknowledgment of a previous request.

The switch input buffer size and memory input queue size were held constant at four for all runs (except in section 3.4). This size was chosen because buffers of size four are easily implementable, and other authors have chosen the same value. Furthermore, we consider only 2×2 switch elements.

[3]This is equivalent to placing an infinite-sized queue between each processor and the first stage of the forward network, and it allows us to precisely control the average packet generation rate. A similar assumption has been made in [12].

3 Performance of the THMM

3.1 The Simulation Models

We consider two machine models. The first is a typical large-scale shared memory multiprocessor, with p PEs connected to p memory units across a $\log p$ stage interconnect, such as the multistage cube network;[2] we refer to this as the Conventional Memory Multiprocessor (CMM). To a close approximation, the CMM also resembles a multiprocessor such as the BBN Butterfly in which the memory in each PE is accessible to any other PE across the interconnection network; this is true because, as in the case of the CMM, each processor has local memory. Accesses by a PE to "nonlocal" memory in the Butterfly are typically to memories in other PEs, but with probability $\frac{1}{p}$ may also be local. The second architecture is the THMM introduced above; the version of the THMM considered consists of p PEs, p global memories, and an interconnect containing a memory hierarchy between them (see figure 1). This interconnect is a $\log p$ stage multistage cube network from the PEs to the memories (the forward network), and a $\log p$ stage multistage cube network from the memories to the PEs (the return network). Each switch stage of the forward network except that closest to the global memories contains memory. Corresponding switches of the forward and return networks of the THMM can be thought of as being connected through their respective memories, as in figure 2. A packet destined for a THMM switch memory traverses the forward network to the appropriate switch and enters the memory queue at that switch; one or more cycles later (depending upon the memory queue length and congestion in the return network), the memory places a packet (corresponding to the value of a read or a write acknowledgment) into the return network; the packet then traverses the return network back to the originating PE.

In each model, we assume that the processors, memories, and switches each have unit cycle time (this assumption is modified in section 3.3). Each network is synchronous and packet switched. Each switch input has a FIFO input queue, as does each memory. If there is no room for a packet in a switch element's input buffer, the packet waits—no packets are lost. The model is realistic, as suggested by Ding and Bhuyan [4], in that no packet may move to the next stage of the interconnect unless there is space in the destination queue at the *beginning* of the cycle.

The stream of memory requests from each processor is independent and identically distributed. Initially, the memory requests are uniformly distributed; in the THMM, the requests to the different levels are also uniformly distributed. In sections 3.5ff., we examine the performance of the THMM with the generation of requests to different levels of the hierarchy drawn from a variety of discrete distributions. The random number generator used throughout is the minimal standard of Park and Miller [16] divided into non-overlapping streams as suggested by Jain [7].

3.2 The Simulation

The principal measure used to compare the performance of the CMM and THMM is the average round-trip delay time of a packet—the number of simulation cycles that elapse as the packet travels from a PE to a specified memory, and then back to the originating PE. This is an important factor in the performance of a shared memory system, since time spent by PEs waiting for communication negatively impacts the speedup. Even in cases where communication and computation are

[2]Since we consider only 2×2 switches in this paper, all logarithms are to the base two.

Such a tree hierarchical architecture could be realized by incorporating memory at each $k \times k$ switching element of a multistage cube network. The idea of incorporating memory at the switching elements has been used to perform barrier synchronization efficiently in hardware [8]; this idea has also been used by Mizrahi, et al. [15] to provide a means for cache coherence within multistage interconnection networks. Except for some similarity in the suggested implementation of the THMM and the architecture proposed by Mizrahi, et al., the scope and nature of the two studies are different. Features of a THMM include:

- The number of PEs that share a memory at level i is k^i.
- The number of nodes at level i visible to each PE is k^{i-1}.
- If a variable occupies a node at level i, then in occupying any other module at level j ($j < i$), it may occupy only those nodes rooted at i. Hence, management of data movement between levels is simple.
- In a typical tree architecture, the root node (the processor or the memory module) may be a performance bottleneck. In a THMM, this problem does not arise since there are effectively p/k root nodes.

A possible implementation of a THMM using 2×2 switching elements is shown in figure 1. We wish to emphasize that a variety of hierarchical architectures are feasible. The THMM is just one example class of hierarchical architectures which we consider to be useful for a wide variety of applications.

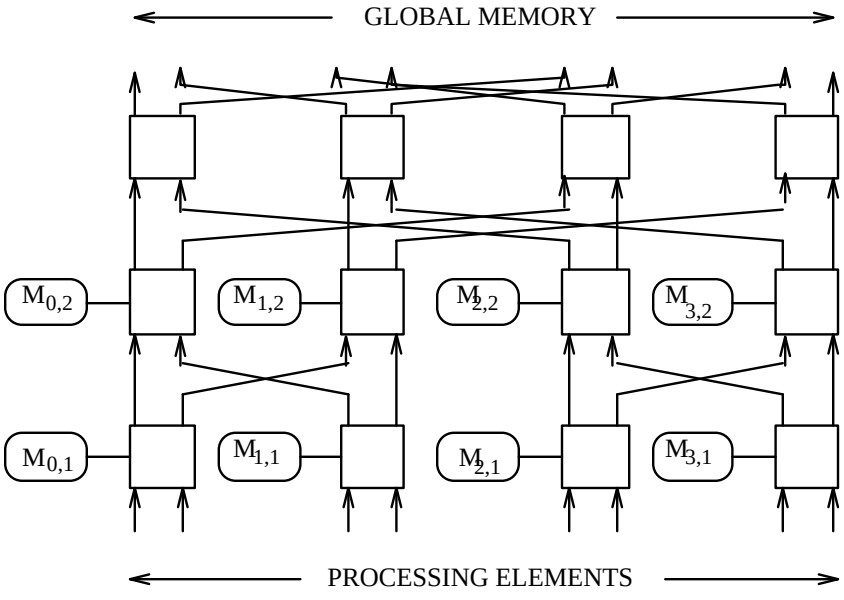

Figure 1: A Possible Implementation of a Tree-Structured Hierarchical Architecture

reduction, barrier synchronization, and prefix computation. For example, during the initial stage of a computation, each PE may interact with its physical neighbor; at a later stage, PEs from a group may interact with another disjoint group of PEs; at a stage after that, PEs from two larger disjoint groups may interact, and so on. Even with a relatively simple cost model which does not account for memory contention, we have found that the running time of these algorithms can be significantly reduced by execution on a hierarchical memory multiprocessor. The reasons to expect better performance *a priori* are twofold. First, the memory latency is reduced. Many memory requests need not travel all the way to the globally shared memory, thus reducing latency. Furthermore, because of reduced traffic to the higher levels of the memory hierarchy, there will be less contention at the switching elements and at the memory modules, further reducing communication delays. Second, synchronization costs are reduced. A large portion of the synchronization cost is often due to many PEs checking a single shared variable while waiting for some asynchronous event to occur. Often, a variable whose degree of sharing changes dynamically over its lifetime can be used; this reduces the number of PEs accessing the variable, and thereby reduces synchronization costs. In fact, in many algorithms, only *a small fixed number of* PEs access any variable in the new architecture. As has been shown analytically in [9], the total latency cost and the total synchronization cost are both significantly reduced in hierarchical architectures.

Given the current state of the art in VLSI technology, are hierarchical memory architectures feasible? With a limited memory at each level and a relatively small number of levels, such architectures are feasible today. We expect this situation to improve in the not too distant future. One possible implementation of a hierarchical memory architecture is to associate memory with each switching element of a multistage interconnection network. Such a scheme will entail an increase in the total width of the data path and simple hardware enhancements to a switching element—for example, to perform the extra routing to the partially shared memory module. We assume that, as in any shared memory multiprocessor, a single address space is used.

We expect a parallelizing compiler to generate code for a hierarchical memory multiprocessor. The parallelizing compiler, along with the memory management system, should exploit the notions of partial sharing; we have investigated compilation of loops with constant-distance dependences in [10].

Another issue is that of the topology of the hierarchical memory architecture. The structure of the parallel algorithm itself imposes a logical hierarchy on the use and update of the variables. A straightforward mapping between this logical hierarchy and the physical hierarchy imposed by the architecture facilitates the calculation of the degree of sharing, the initial placement of the data at its appropriate memory level by the compiler, and the dynamic movement of the data by the memory management system. A class of hierarchical architectures which we regard to be particularly useful is considered in the next subsection.

2.1 Tree-Hierarchical Architectures

In a hierarchical memory architecture, each memory can be assigned a level number. The memory at the lowest level (the local memory) is level zero, the next higher level is level one, and so on. Each memory at level i could be made up of multiple memory modules or nodes. We number the modules and the PEs left to right beginning with zero. Let the set of all PEs to which the variables in node i at level j of the hierarchy are visible be denoted by P_{ij}. In tree architectures such as the THMM:

$$P_{ij} = P_{kj} \quad or \quad P_{ij} \cap P_{kj} = \phi, \quad (i \neq k)$$

1 Introduction

It is well known that parallel processing offers a promising future to achieve high computational speeds. Parallel processing systems with hundreds or even thousands of processing elements (PEs) are now feasible. In shared memory multiprocessing based on the von Neumann model, two fundamental problems—the communication overhead arising from long memory latency and the overhead arising from the synchronization required among the subtasks of a parallel program—reduce speedup, and consequently defeat the very purpose of parallel processing (for a detailed discussion, see [3]). Typically, in scalable shared memory multiprocessors, the shared memory is organized as a set of modules connected to the PEs via a multistage interconnection network [5]. These modules could be physically separate from the PEs, as in the University of Illinois Cedar and the NYU Ultracomputer multiprocessors, or a part of the memory associated locally with each PE, as in the BBN Butterfly multiprocessor (for a discussion of the architectural features of the Butterfly, Cedar, and the Ultracomputer see [2]).

The use of local memories and caches have been among several ways of reducing latency delays. Only global read-only and local read/write variables can, however, be stored in the local memory. Use of caches gives rise to the problem of cache coherence which invariably leads to higher synchronization costs [3].[1] Combining [2] reduces contention but does not reduce the portion of the latency due to the time required for each packet to travel through the $\log p$ stages of the interconnect.

To reduce synchronization overheads, memory modules with simple hardware enhancements such as adders and comparators have been proposed in the design of the Ultracomputer and Cedar architectures. Such a solution does not directly reduce the number of communications and is also *ad hoc* since it does not scale with the size of the multiprocessor.

In the next section we discuss an alternative architecture, the *Tree-structured Hierarchical Memory Multiprocessor* (THMM), which reduces the latency and the synchronization delays mentioned above. This architecture has been previously discussed in [9, 14]. Central to the efficient use of the THMM is the notion of *partially shared variables* which provides a systematic way of utilizing the hierarchical memory [9]. Section 3 consists of a performance comparison of the THMM with a conventional shared memory multiprocessor; this comparison is based upon a stochastic simulation of the two architectures. Section 4 gives a brief discussion of possible future research directions.

2 The Architecture

A memory hierarchy is a set of partially shared memories such that a variable at level i of the hierarchy can be shared by more PEs than a variable at level j, where $j < i$. An architecture incorporating a memory hierarchy has the local memory at its lowest level and the fully-shared global memory at its highest level, with a "continuum" in between. Traditional multiprocessor architectures incorporate a one or two level hierarchy. The Cedar multiprocessor incorporates a three level hierarchy. Other researchers have studied ways to exploit two or three level hierarchies [6, 13]. A theoretical model of hierarchical memory for a uniprocessor system has been investigated in [1].

Many parallel algorithms require interaction among the PEs in a structured hierarchical manner; these include algorithms for matrix multiplication, merge sort, five point differencing, vector

[1] The use of caches is orthogonal to the use of hierarchical shared memory. Hence, caches could be implemented along with hierarchical shared memory architectures [9].

Abstract

Latency and synchronization overheads have been identified as fundamental problems in large-scale shared memory multiprocessing. In an earlier paper, one of the authors introduced a hierarchical shared memory multiprocessor architecture [9]. This architecture, based on hierarchical shared memory, exploits the notion of *partial sharing* of variables to reduce latency and synchronization costs. Using a stochastic model for generation of memory requests by each processor, a simulation-based study of the performance of the Tree-structured Hierarchical Memory Multiprocessor architecture (THMM) is compared to that of a conventional shared memory multiprocessor architecture (CMM). It is found that under a variety of stochastically generated workloads, the THMM outperforms a CMM with the same number of processors.

Key words: hierarchical shared memory multiprocessing, latency, contention, simulation.

Performance of a Tree-Structured Hierarchical Memory Multiprocessor

J. D. Martens
D. N. Jayasimha*
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210
martens@cis.ohio-state.edu
jayasim@cis.ohio-state.edu

9 August 1992

*Both authors were supported in part by the National Science Foundation under Grant No. NSF CCR-8909189 and by The Ohio State University Office of Research and Graduate Studies under Grant No. 221146.