PROCESSOR ASSIGNMENT AND SYNCHRONIZATION IN PARALLEL SIMULATION OF MULTISTAGE INTERCONNECTION NETWORKS

P. Goli, P. Heidelberger, D. Towsley, Q. Yu

COINS Technical Report 89-78

Processor Assignment and Synchronization in Parallel Simulation of Multistage Interconnection Networks

P. Goli*, P. Heidelberger**, D. Towsley***, Q. Yu*** [1]

*Department of Electrical and Computer Engineering
University of Massachusetts
Amherst, Massachusetts 01003

**IBM Research Division
T.J.Watson Research Center, Hawthorne
P.O. Box 704
Yorktown Heights, New York 10598

***Department of Computer and Information Sciences
University of Massachusetts
Amherst, Massachusetts 01003

Abstract

Multistage interconnection networks (MINs), an important class of networks arising in parallel computer architectures, are excellent candidates for parallel time-driven simulation on shared memory computers because they are large, inherently parallel, regularly structured, discrete time systems. In this paper we report results from a continuing study on the performance of such simulations on a Sequent Symmetry. Our focus is the class of buffered delta networks consisting of 2×2 switches. We report results for simulations of MINs containing 4–9 stages and report on the effects that different synchronization techniques, different processor to switch allocation strategies, and non-uniform traffic patterns in the workload have on simulation speedup. Briefly, we observe that a two-phase synchronization technique requiring only two barriers during each clock cycle provides the best speedup. We examine two processor to switch allocation strategies, a *contiguous* allocation that allocates contiguous rows of switches to each processor, and an *interleaved* allocation that allocates every P-th row to each of the P processors. While the contiguous allocation exhibits better locality of reference, for uniform traffic there is little difference in performance between the two strategies. For non-uniform traffic, whereas the speedup using contiguous allocation degrades, the speedup using the interleaved allocation remains nearly constant. Using these techniques, speedups of over 14 on a 16 processor system have been obtained.

[1]The work of the first and third authors was supported by a joint research contract from IBM and performed on equipment provided by the National Science Foundation under grant CCR-8500332. The work of the fourth author was supported in part by the National Science Foundation under grant CCR-8712410.

1 Introduction

Simulations of large scale queueing networks are often computationally expensive and the parallelization of such simulations is currently an active area of research (see, e.g., [10,11] and the references therein). A variety of different algorithms have been proposed to parallelize the simulation, most notably Jefferson and Sowizral's "Time Warp" algorithm which is a so-called optimistic mechanism [3] and a so-called conservative algorithm proposed by Chandy and Misra [1]. Performance of these algorithms for queueing network simulations is reported in [2,4,9,10,11] . These algorithms show promise when applied to special classes of networks (see, e.g., [2,4]).

The above mentioned algorithms are event-driven in the sense that a processor advances its simulation clock to the time of the next event scheduled at the queues managed by that processor. We have been exploring time-driven simulation as an alternative approach (see Peacock, Wong and Manning [7]) in which there is a fixed, constant time increment. In this approach, processors simulate only those events scheduled at the current time step and then synchronize before advancing to the next time step. In an earlier paper [12] we presented some preliminary results from a study of parallel time-driven simulation of Multistage Interconnection Networks (MINs) on a Sequent Symmetry. MINs are an important class of networks arising in a number of highly parallel shared memory computer architectures. In [12], we argued why MINs are ideal for time-driven parallel simulation: they have a highly regular structure and are naturally discrete time systems with many events occurring at each time step. We considered three synchronization methods for synchronizing the processors during each time step and observed that a multibarrier method performed the best. The multibarrier method requires the processors to perform a barrier after simulating its allocated switches at each stage (except the first). All of the results were for eight processors and uniform traffic patterns.

In this paper we present additional results from our continuing study of parallel time-driven simulation of buffered Delta MINs, focusing on improved processor allocation and synchronization algorithms which take advantage of the network's topological structure. This structure can be exploited to improve locality of reference, reduce the number of synchronization barriers, and to equalize the processor workload under non-uniform traffic conditions. For example, an improved multibarrier method can be devised which reduces the number of barriers per cycle from $(n-1)$ to $\log P$, where n is the number of stages in the network and P is the number of processors. At little additional computational cost, the number of barriers can further be reduced to two per cycle using a *two-phase* method. In this approach, processors simulate the odd numbered stages in one phase, perform a barrier, then simulate the even numbered stages and perform a second barrier. We have achieved speedups of over 14 on a 16 processor system using the two-phase approach.

We also consider the effect that non-uniform traffic in the form of "hot spots" have on speedup. As described in [8], the hot spot model is important architecturally and the performance of buffered MINs degrades severely in the presence of hot spots. We observe that there is a noticeable decrease in speedup as the traffic becomes non-uniform. In these and previous simulations we assign contiguous rows of switches to each processor. However, when the traffic load exhibits hot spots, this places an unequal computational load on some processors. Hence, we also examine the performance of a second assignment rule, an interleaved assignment, where rows are assigned to processors mod P. This assignment rule tends to even out non-uniformities in processor workloads. Similar types of assignment rules have been used in parallel computation for different problem domains (see, e.g., [5]). While the contiguous allocation exhibits better locality of reference than the interleaved allocation, there is little difference in the speedups using these allocations for uniform traffic. However, in the case of hot spots, the speedup decreases much more using the contiguous allocation. In fact, the speedup using the interleaved allocation remains nearly constant over the range of interest.

This paper is organized as follows. We briefly describe delta networks in the next section. Section 3 contains descriptions of the different approaches to synchronizing the processors and to assigning switches to processors during the simulations. Section 4 provides details of the simulator. The experiments and their results are reported in Section 5. Section 6 summarizes the results of our study.

2 Delta Networks

We are interested in a class of MINs called *buffered delta networks* composed of 2 input, 2 output switches. An n stage delta network contains 2^{n-1} switches at each stage and can be used to connect 2^n inputs to 2^n outputs. The interconnection pattern of links between output ports in one stage and the input ports of the next stage is such that there is exactly one path between any one of the input ports at the first stage and any one of the output ports at the last stage. Buffers for storing packets are present at each input port at each switch. We consider two classes of delta networks, *single buffer delta networks* where each input port can store at most one packet and *infinite buffer delta networks* where any number of packets can be stored at an input port. Figure 1 shows a delta network with 4 stages.

We consider the subclass of *synchronous* delta networks. By synchronous, we mean that time is divided into fixed length intervals and that the actions taken by a switch to select a packet and transmit that packet to the next stage occur in one of these time intervals. We assume that each of these time intervals is exactly one unit of time long (the time unit is called a cycle). If two packets arrive at a switch simultaneously, both of them can be transmitted provided they are directed to different output ports. Otherwise one packet is selected randomly and transmitted; the other remains in its buffer. We have studied two different single buffer implementations:

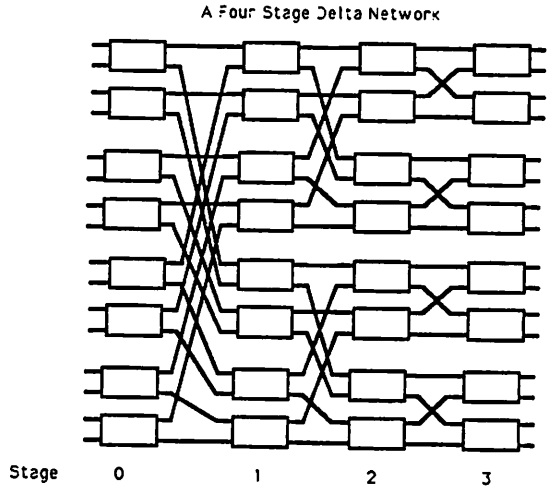

Figure 1: A Four Stage Delta Network.

Type I : In this implementation a message may transfer from one stage to another provided that there is space in the buffer *by the end of the clock cycle.* Hence, transfers may take place even though the buffer may be full at the beginning. Transfers between two stages, under this scheme, may depend on the buffer occupancies of stages far downstream.

Type II : In the second implementation, a message may transfer from one stage to another *only* if there is space in the buffer at the beginning of the clock cycle. The hardware for this type of single buffer switch is easier to build. It also has the property that a transfer between two stages is determined only by the buffer occupancy of the next downstream stage. Hence, it is easier to simulate under different synchronization methods.

Our earlier paper [12], reported results on Type I single buffer MIN's. This paper reports results on Type II single buffer MIN's.

Our studies assume that arrivals to each input are described by a Bernoulli process and that the processes are independent and statistically identical from input to input. We consider two scenarios. In the first scenario, each packet is destined for each output with equal probability; we refer to this as the *uniform traffic* model. We also consider a non-uniform traffic pattern where each packet chooses the output with address 0 with probability p_h and each of the other outputs with equal probability $(1 - p_h)/(N - 1)$, where N is the total number of outputs. In order to parameterize p_h, we define a parameter f, such that $p_h = f/N$. In our simulations,

f takes values 1,3,5 and 10 (f=1 corresponds to the uniform traffic model). This is the so called "hot spot" model, with output number 0 representing a heavily utilized, or hot, memory module in a model of a shared memory multiprocessor. Last, we are interested in determining the average packet delay and the average buffer occupancies. The parameter λ denotes the packet arrival rate at each input port, i.e., on each cycle a new packet is generated at each input with probability λ.

We conclude this section with some additional notation to be used later. A switch in the network is denoted as an element in a matrix. $s_{i,j}$ represents the switch on row i at stage j, $0 \leq i \leq 2^{n-1}$, $0 \leq j \leq n-1$. Stage 0 is the one with the inputs to the MIN and stage $n-1$ is the one with outputs from the MIN. They are sometimes referred to as the first stage and the last stage.

3 The Sequent Symmetry System

The Sequent Symmetry system at the University of Massachusetts has 16 Intel 80386 processors running at 16 MHz and 128 Mbytes of fully interleaved physical memory. Each processor contains 64 Kbytes, 2-way set associative cache and a floating-point coprocessor. These cache memories optimize system performance both by limiting traffic on the system bus and by improving individual processor performance.

The Symmetry cache memory uses an ownership-based copy-back caching policy to maintain consistency between copies of data in main memory and the local processor caches while limiting traffic on the system bus. When a processor modifies a piece of data in its cache, it does not write the new value back to the system memory or to other processors caches until another processor needs that data or until it needs the cache space occupied by that data. The processor caches enforce the caching policy by communicating with each other through a set of signals and through a set of status tags associated with each piece of data. The reader is referred to [6] for further details. The results reported in in this paper are from a Sequent Symmetry that has been upgraded since the time that the experiments reported in [12] were run. Hence the two sets of results are not directly comparable.

Sequent computers run the DYNIX system, a version of UNIXbsd with extensions for processor scheduling and other multiprocessing support. Higher level synchronization primitives such as *barriers* are provided and are used in our simulations. Barrier is a synchronization point for a group of processes. A process must wait at the barrier until all the processes in the group are present at the barrier.

4 Processor to Switch Assignment and Processor Synchronization

An n stage MIN has 2^{n-1} rows of switches. Let the number of processors, P, be expressible as a power of 2, $P = 2^m$ where $m < n$. The simulation is time-driven. During each time unit, a processor simulates the 2^{n-m-1} switches assigned to it at stage n. In order for the simulation to be correct, it must handle the following constraints:

mutual exclusion - During a parallel simulation of a MIN, a buffer may be accessed by two different processors, one simulating the switch that feeds packets into the buffer and the other simulating the switch to which the buffer belongs. Only one of these processors can access the buffer at a time.

single passage - It is our assumption that selecting and passing a packet takes one time unit (cycle). In the simulation, a packet can be passed at most once during one cycle. That is, if switch $s_{i,j}$ passes a packet to switch $s_{i_1,j+1}$, this packet should not be passed by $s_{i_1,j+1}$ to a switch at stage $j+2$ until the next cycle (at the earliest).

We allocate switches to processors in two different ways. If we label the processors $0, 1, \ldots 2^m - 1$, then, in the first allocation we assign the switches in rows $(j2^{m-n-1}+1, \ldots, (j+1)2^{m-n-1})$ are assigned to the j-th processor. This assignment is called *contiguous* allocation. In the second scheme, called the *interleaved* allocation scheme, row i is assigned to the processor labeled $i \bmod P$, where P is the number of processors.

In order to ensure that the clocks are synchronized at the processors, we require the processors to execute at least one barrier during each unit of time. However, a single barrier is not sufficient to ensure that the above constraints are satisfied.

We describe two different approaches to handle synchronization.

Multibarrier: Multiple barriers are used to synchronize the processors. Two schemes are considered which differ only in the number of barriers required. In the first scheme $(n-1)$ barriers are used for simulating a network with n stages. All the processors execute a barrier between simulating switches at two adjacent stages. Since the switches in the first and last stages never communicate directly with one another, one barrier suffices for both the first and the last stage. With such a strict synchronization method it is impossible for two processors to simulate switches at different intermediate stages simultaneously. Hence, mutual exclusion is preserved. Single passage is guaranteed by simulating switches from the last stage to the first stage. Both interleaved and contiguous allocation can be used with this approach.

In the second scheme, $\log P$ barriers are used for simulating a network with n stages and using P processors. Here we exploit the topology of the delta networks. Delta networks have blocks

of $(n-m) \times 2^{n-m-1}$ switches that do not share any links with other similar sized blocks in the last $(n-m)$ stages, where $m = \log P$. The switches in a block are assigned to a single processor. No synchronization is required within the block since the whole block is assigned to a single processor. Only the contiguous allocation scheme can be used here. To see this, consider Figure 1 with $P = 4$. The contiguous allocation assigns processor 0 to ports 0 through 3, processor 1 to ports 4 through 7, etc. Clearly, processors can simulate the last two stages independently.

Two-Phase: All of the processors divide up the simulation of one clock cycle into two phases. In the first phase, referred to as the *odd-phase*, the switches in the odd numbered stages $(1, 3, \ldots)$ are simulated. The processors synchronize and then execute a second phase, the *even-phase*, where they simulate the switches in the even numbered stages $(0, 2, \ldots)$. The processes synchronize again and the cycle is repeated. Thus, only two barriers are required. This scheme ensures mutual exclusion because two processors at a time cannot be simulating adjacent stages. To ensure single passage, a timestamp is passed with each packet. The timestamp indicates the cycle in which the packet has arrived at a particular stage. Each processor checks the timestamp on the packet when simulating a switch. If the timestamp is less than the current cycle, the packet is eligible to be passed on to the next stage. If the timestamp is equal to the current cycle, the packet is not processed in the current cycle. Both contiguous and interleaved allocation can be used with this method of synchronization.

5 Simulator Details

In an infinite buffer network, for multibarrier simulations, a packet is composed of three fields: *arrival_time*, *destination_address*, and *pointer_to_next_packet*. The *arrival_time* is used for calculating the response time, the *destination_address* is used for determining the packet routing, and the *pointer_to_next_packet* is used to maintain the queues in the buffers. In the case of two-phase synchronization scheme, a fourth field, *timestamp* is added to the packet to indicate the cycle in which a packet has arrived at a particular stage. For single-buffer networks, the *pointer_to_next_packet* field is not required.

A packet is created and introduced into the network at stage 0. It is then routed through the network of switches and is finally removed from the network at the last stage. FCFS queueing discipline is used at each switch. Both the serial and parallel simulators are written in C. The parallel simulator is essentially the same as the serial simulator except for the additional parallel program primitives for forking and synchronizing the processes. A packet is declared to be *shared* so as to allow access to it by all the processors. In the Sequent Symmetry, a shared data item remains in a processor's local cache and is not written into the system memory until the processor needs the cache space occupied by that data or until another processor needs that data. Additional time is spent if a processor needs to access a data item that is present in

another processor's local cache or in the system memory.

Two switches in adjacent stages are said to be *adjacent* if there is a link connecting them. A packet is routed from input to output through switches that are adjacent. A link is said to be *shared* if it connects two adjacent switches that are simulated by two different processors. A packet remains in a processor's local cache as long as it is routed through adjacent switches that are simulated by the same processor, i.e., through non-shared links. When a packet is routed over a shared link, the packet is moved from one processor's local cache to another processor's local cache. Tables 1(a) and 1(b) list the number of shared links at input and output per stage per processor for a 9 stage MIN simulated using 8 processors for contiguous and interleaved allocations, respectively. The contiguous allocation has many fewer shared links than the interleaved allocation. Thus, we expect simulations using the contiguous allocation to have higher cache hit ratios, although the Sequent is not instrumented to measure this quantity. Table 1(a) also helps explain the $\log P$ multibarrier synchronization method: there are no shared links in stages 4 through 8.

Simulation of a switch by a processor involves removing the packets from the two input ports of the switch, computing the output port address, and placing the packets into the input buffers of the switches in the next stage. When the packets at both the inputs are contending for the same output port, one of them is selected randomly.

6 Results

Execution times of the time-driven parallel simulation on 8 and 16 processors are compared with the execution times of the serial simulation. The execution times we used in the speedup calculations include model initialization, process forking, and process termination.

6.1 Uniform Traffic

Simulations were conducted on the infinite buffer delta network. The number of stages varies from 4 to 9 and the arrival rate, λ, takes values 1/4, 1/2, and 3/4. We ran each simulation for 10,000 time units.

Figure 2 compares the speedups obtained with 8 processors and multibarrier synchronization, with $(n-1)$ barriers, $\log P$ barriers and the two-phase mechanism. The contiguous allocation scheme is used in all the schemes. The speedup obtained due to the $\log P$ barrier approach is higher than the speedup obtained for the $(n-1)$ barrier approach. When the number of stages is equal to 4 and with 8 processors, the number of barriers used in both cases is the same. The speedup obtained with the two-phase mechanism is higher than the multibarrier in all cases. This is due to the reduced number of barriers (two) in the two-phase scheme.

Stage	Processor							
	0,1,3,7		2		4,5		6	
	i/p	o/p	i/p	o/p	i/p	o/p	i/p	o/p
0	0	32	0	32	0	64	0	64
1	32	32	32	64	64	32	64	64
2	32	32	64	32	32	32	64	32
3	32	0	32	0	32	0	32	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0

(a)

Stage	Processor									
	0,7		1,3		2		4,5		6	
	i/p	o/p	i/p	o/p	i/p	o/p	i/p	o/p	i/p	o/p
0	0	16	0	32	0	32	0	32	0	32
1	16	16	32	32	32	32	32	32	32	32
2	16	16	32	32	32	32	32	32	32	32
3	16	16	32	32	32	32	32	32	32	32
4	16	16	32	32	32	32	32	32	32	32
5	16	32	32	32	32	32	32	64	32	64
6	32	32	32	32	32	64	64	32	64	64
7	32	32	32	32	64	32	32	32	64	32
8	32	0	32	0	32	0	32	0	32	0

(b)

Table 1: Shared Links for: (a) Contiguous Allocation, (b) for Interleaved Allocation

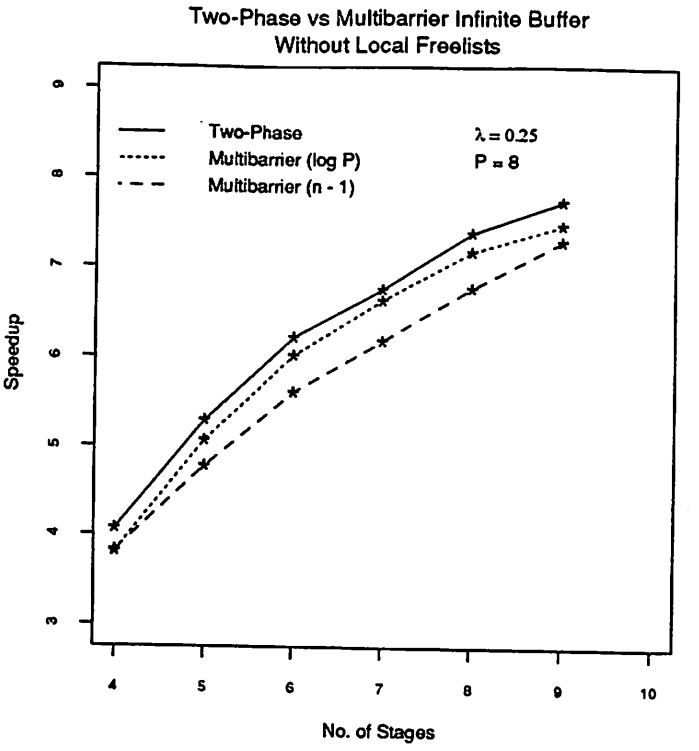

Figure 2: Comparison of Synchronization

There are two kinds of penalties involved when a barrier is executed: the time for executing the barrier statement, and secondly the time penalty incurred while waiting for synchronization at the barrier.

Figure 3 compares the speedups when 16 processors are used for parallel simulation. Notice that the speedups obtained are not as encouraging as they are for the 8 processor simulations. This motivated us to look closely at the computation time for each stage. We observed that unlike the 8 processor aimuation, the 16 processor simulation spends a large fraction of its time in the first and the last stages. In these stages, a packet is created and destroyed by allocating and deallocating memory using a C routine. We conjectured that the poor performance is due to contention among the processors for the bus and for memory in the global address space while executing these routines. We modified the simulator so that each processor maintains its own *freelist* from which it attempts to allocate and deallocate memory when creating and destroying packets. When a packet reaches the destination, the memory space allocated to that packet is added to the freelist rather than returning to the system memory. Its own local freelist is used to allocate memory for any new packets that are created in the first stage by that processor. Only when the local freelist is empty will the processor execute the routine to allocate the memory. In order to make a fair comparison, we also modified also the serial simulator to maintain the freelist. Figure 4 compares the speedups obtained for 16 processor simulations using the two-phase mechanism with and without local freelists. Clearly, the two-phase mechanism which maintains the freelist is superior. We point out that the absolute time for the serial simulation

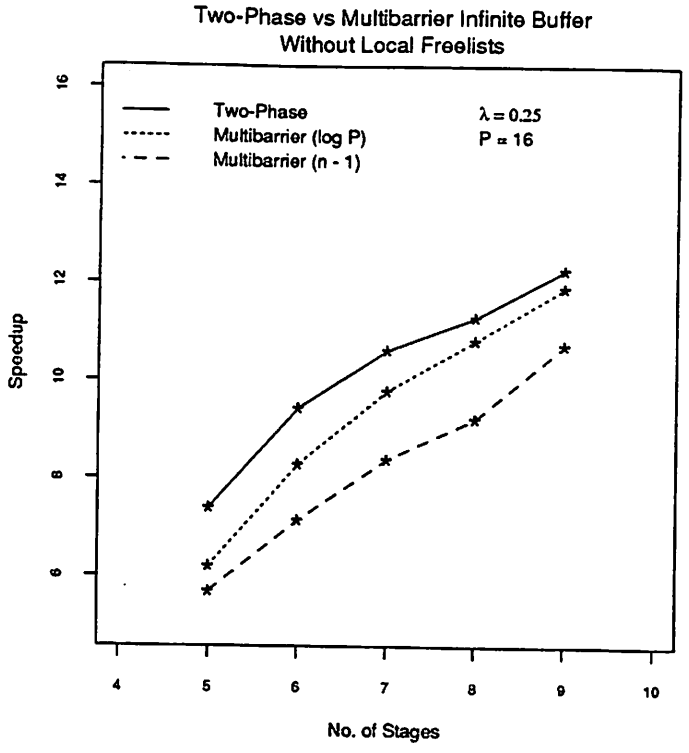

Figure 3: Comparison of Two-Phase and Multibarrier Infinite Buffer

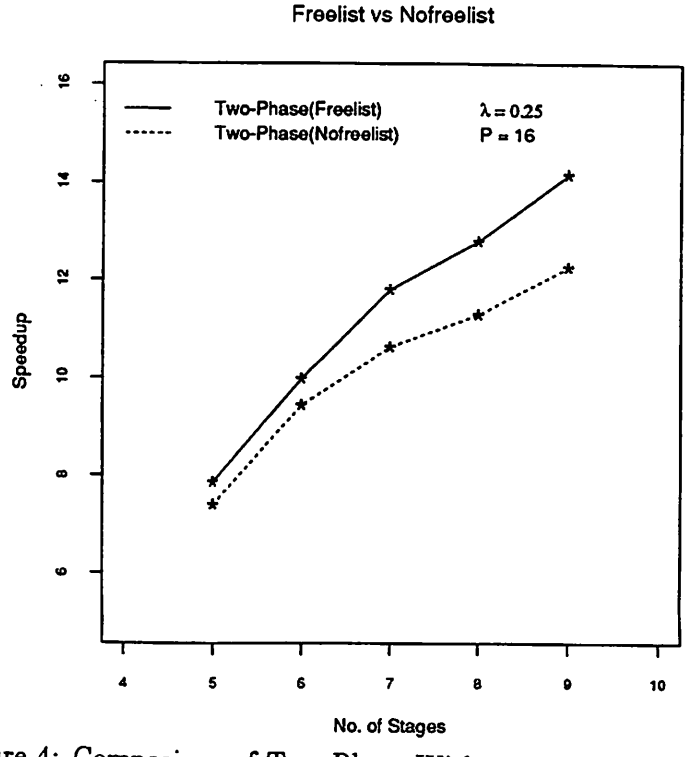

Figure 4: Comparison of Two-Phase With and Without Local Freelists

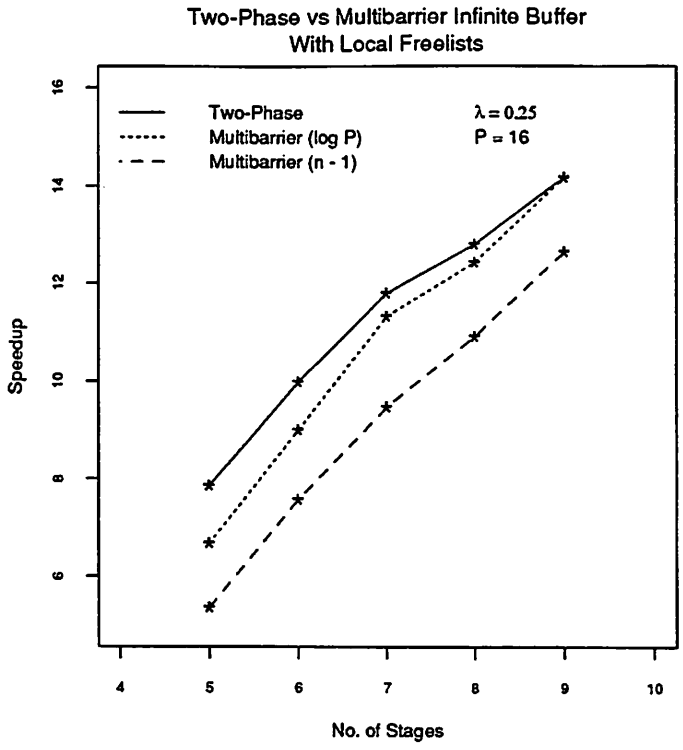

Figure 5: Comparison of Two-Phase and Multibarrier using Local Freelists

also decreased when a freelist was maintained. Similar improvements were observed for the multibarrier mechanism for 8 and 16 processors. The two-phase mechanism and multibarrier mechanism with $(n-1)$ and $\log P$ barriers using local freelists are compared in Figure 5.

6.2 Non-Uniform Traffic

We now consider the effect of non-uniform traffic on the speedups obtained from parallel simulations using different processor-switch assignments. We use the hot spot model described earlier. As described in [8], since a port can output at most one packet per cycle, contention due to the hot spot causes the interconnection network to saturate in tree-like regions. The hot output port can process only one packet per cycle, so its predecessors become backed up. Since the predecessors cannot output more than one packet per cycle, the predecessors of predecessor saturate, and so on. This tree-like saturation behavior suggests that processors will not have a balanced load under the contiguous allocation. This motivated us to consider a different allocation strategy which would better balance the load among the processors. Hence, we considered the *interleaved* allocation scheme. Table 2 lists the number of switches that encounter hot spot traffic per stage and the number of processors that simulate these switches using the contiguous and interleaved allocation schemes for a 9 stage MIN simulated using 8 processors. From Table 2, the interleaved allocation clearly provides better workload balancing, particularly in the middle stages of the network. This will be true regardless of where a single hot spot is located,

and this allocation will also tend to equalize the workload for more general non-uniform access patterns, e.g., multiple hot spots.

Stage Number	No of Hot Switches	Contiguous Allocation	Interleaved Allocation
0	256	8	8
1	128	4	8
2	64	2	8
3	32	1	8
4	16	1	8
5	8	1	8
6	4	1	4
7	2	1	2
8	1	1	1

Table 2: Distribution of Hot Spot Traffic Load

The speedups obtained from the interleaved allocation are compared with the speedups obtained from the contiguous allocation in Figure 6 for the single buffer delta network with Type II switches. We plot the speedup for both the two-phase and log P multibarrier synchronization methods in the case of the contiguous allocation. However, only the two-phase synchronization method is implementable with the interleaved allocation. Results are plotted for 6 and 9 stages. For uniform traffic, $f = 1$, the speedup using the two allocations is nearly the same. The interleaved allocation performs better as the hot spot parameter f increases. Even though there are greater number of shared links in the interleaved allocation, the performance is better because of the better workload balancing. In the contiguous assignment approach, the computation load is not evenly distributed and the less busy processors need to wait longer for synchronization at the barrier. Notice that the speedup for the interleaved allocation remains almost constant as the hot spot traffic increases, whereas the speedup for the contiguous allocation decreases as the hot spot traffic is increased.

7 Summary

In this paper, we presented results from a continuing study of time-driven parallel simulation of delta networks on a shared memory system. We examined several processor synchronization and allocation rules. We found that a two-phase synchronization approach, requiring only two barriers per cycle, provides the best speedup. Using an interleaved processor to switch allocation scheme, the special structure of delta networks can further be exploited to provide balanced

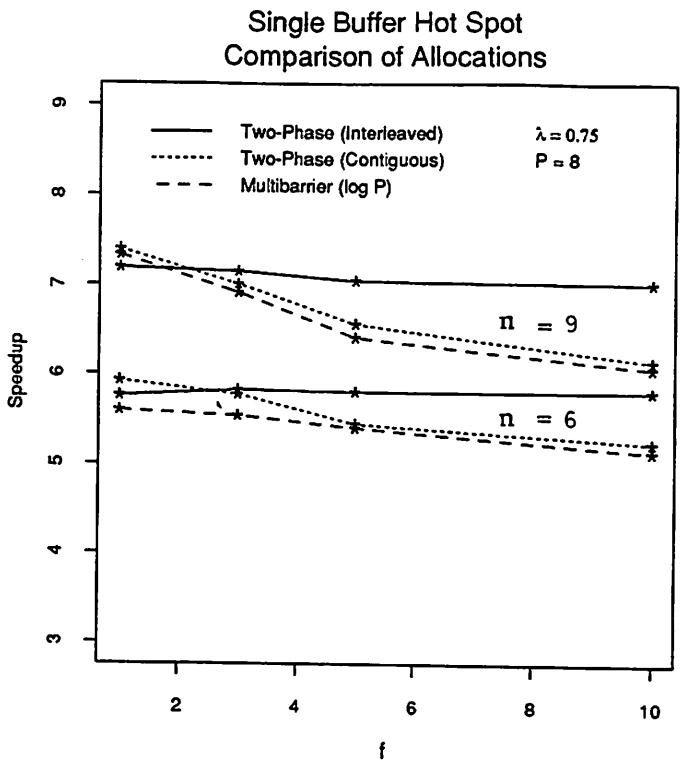

Figure 6: Comparison of Allocations

processor workloads under certain non-uniform traffic patterns of interest. Once a memory allocation bottleneck was removed from our simulator, we were able to achieve nearly all of the available parallelism on large problems, obtaining speedups of over 14 on a 16 processor system.

While we considered only buffered delta networks with 2×2 switches , these techniques extend easily to other types of MINs operating in discrete time, e.g., multi-path, or unbuffered networks. We expect that similar success can be obtained for such MINs as well. The networks considered here are of a special structure, yet they are extremely important both in computer architecture and in communications systems. In addition, simulation of such networks is computationally expensive. We have demonstrated that a simple time-driven parallel simulation approach is highly effective and can be used *today* to study architectural issues. Since computer systems are inherently discrete time systems, we believe that the parallel time-driven simulation approach will also be applicable to other areas of computer architecture.

References

[1] K.M. Chandy and J. Misra, "Asynchronous Distributed Simulation via a Sequence of Parallel Computations," *CACM*, Vol. 24, No. 3, (April 1981), 198–206.

[2] Richard M. Fujimoto, "Time Warp on a Shared Memory Multiprocessor," to appear in *Proceedings of the International Conference on Parallel Processing*, August 1989.

[3] D.R. Jefferson, "Virtual Time," *ACM Transactions on Programming Languages and Systems*, Vol. 7, No. 3, (July 1985), 404–425.

[4] D.M. Nicol, "Parallel Discrete-Event Simulation of FCFS Stochastic Queueing Networks," *Parallel Programming: Experience with Applications, Languages, and Systems*, ACM SIGPLAN (July 1988), 124–137.

[5] D.M.Nicol, "Dynamic Remapping of Parallel Time-Stepped Simulations," *Distributed Simulation*, 1989. B. Unger, R. Fujimoto (editors). The Society for Computer Simulation International, San Diego (1989), 121–125.

[6] *Symmetry Technical Summary*, Sequent Computer Systems, Inc.

[7] J.K. Peacock, J.W. Wong and E. Manning, "Distributed Simulation Using a Network of Processors," *Computer Networks*, Vol. 3, (1979), 44–56.

[8] G.F. Pfister and V.A. Norton, "Hot Spot" Contention and Combining in Multistage Interconnection Networks," *IEEE Trans. on Computers*, Vol. 34, No. 10, (October 1985), 943–948.

[9] D.A. Reed, A.D. Malony and B.D. McCredie, "Parallel Discrete Event Simulation Using Shared Memory," *IEEE Trans. on Soft. Engg.*, Vol. 14, No. 4, (April 1988), 541–553.

[10] B. Unger and R. Fujimoto (editors), *Distributed Simulation, 1989*. The Society for Computer Simulation International, San Diego (1989).

[11] B. Unger and D. Jefferson (editors), *Distributed Simulation*. The Society for Computer Simulation International, San Diego (1988).

[12] Q. Yu, D. Towsley, P. Heidelberger, "Time-driven Parallel Simulation of Multistage Interconnection Networks," *Distributed Simulation*, 1989. B. Unger, R. Fujimoto (editors). The Society of Computer Simulation International, San Diego (1989), 191–196.