TIME-DRIVEN PARALLEL SIMULATION OF MULTISTAGE INTERCONNECTION NETWORKS

Qing Yu, Don Towsley, Philip Heidelberger

COINS Technical Report 88-79

August 16, 1988

Time-Driven Parallel Simulation of Multistage Interconnection Networks

Qing Yu and Don Towsley [1]
Department of Computer and Information Science
University of Massachusetts
Amherst, MA 01003

Philip Heidelberger
IBM Research Division
T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

August 16, 1988

Abstract

Multistage interconnection networks (MINs), an important class of networks arising in highly parallel computer systems, are excellent candidates for parallel time-driven simulations on shared memory computers because: 1) they are naturally discrete time systems, 2) they (and hence their models) are inherently parallel and many packets are processed in parallel, and 3) it is possible to exploit their topological regularity. In this paper we report results on the performance of such simulations on an 8 processor Sequent Symmetry system. We simulate both infinite buffer and finite buffer MINs. In the former case we consider several mechanisms to synchronize the processors and find that a simple multibarrier mechanism that requires all processors to synchronize after simulating each MIN stage provides the best performance. We observe that traffic load has little effect on speedup. On the other hand the size of the MIN being simulated has a significant effect on speedup. Speedups typically range from 4 when the MIN has 4 stages to just over 7 when the MIN has 9 stages.

[1]The work of the first two authors was supported in part by the National Science Foundation under grants CCR-8712410, CCR-8500332 and the Office of Naval Research under contract N00014-87-0796.

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., Unger and Jefferson [16] and the references therein). The problem of parallelizing the simulation is analogous to the problem of database consistency in a distributed database system; namely ensuring that simulation events appear to be processed in the correct order (actually the requirement is more stringent than transaction serializability since only a particular serial order is correct). 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 [6] and a so-called conservative algorithm proposed by Chandy and Misra [3]. Performance of these algorithms for queueing network simulations are reported in [5,8,14,16]. These algorithms show some promise when applied to special classes of networks. For example, Nicol [8] has achieved speedups of approximately 11 on a 16 processor system by applying an optimized version of the conservative algorithm to FCFS networks (a FCFS network is a queueing network in which all queueing disciplines are first come first served).

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. An alternative approach is "time-driven" simulation (see Peacock, Wong and Manning [10]) 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. This approach has largely been discarded for queueing simulations since it is thought that only a few events are likely to be scheduled at any given time step. It is worth noting that the time-driven approach is analogous to the numerical solution of differential equations as well as to logic simulation. In fact, parallel processing has been extremely effective for logic simulation and special purpose parallel hardware has been built for this purpose, [13].

Because, at a low enough level, computer systems are inherently discrete time systems having a fixed, constant cycle time, it is possible to identify an important class of computer system models which is a potentially attractive candidate for parallel time-driven simulation: Multistage Interconnection Networks (MINs) [2,4]. Such networks are the foundation of a number of highly parallel shared memory machine architectures [11,15]. They are used to connect a large number of processors to each other or to a

similar number of memories.

The queueing network models of such networks typically do not have analytically tractable solutions and simulations of MINs on a single processor can be very costly. Since the number of switches in a MIN for an N processor system grows like $N \log(N)$, we expect the simulation execution times to grow faster than N. With the move to ever greater levels of parallelism, we expect the computational requirements of simulating highly parallel systems and their associated MINs to strain the capabilities of uniprocessors.

MINs are excellent candidates for parallel time-driven simulations on shared memory computers for the following reasons:

- MINs are naturally discrete time systems with a common, fixed time increment (called the cycle time).
- MINs (and hence their models) are inherently parallel and many packets are processed in parallel. Unless the traffic is low, many switches will be active on each cycle thereby presenting a potentially high level of parallelism.
- It is possible to exploit the topological regularity of MINs. By proper assignment of switches to processors, processors need communicate and synchronize with only a few other processors.

In this paper we report results on the performance of parallel time-driven simulation of MINs on an 8 processor Sequent Symmetry system. We focus on a particular class of MINs, buffered delta networks consisting of 2×2 switches. We consider MINs in which there are infinite size buffers at the switch inputs and in which there is buffer space for at most one packet at each switch input. In the case of infinite size buffers, packets can move forward regardless of the buffer occupancy at the downstream switches. Thi simplifies the problem of synchronizing processors during a simulation and we report on three different synchronization methods. In the case of finite buffers, whether or not a packet can be transmitted depends on the buffer occupancy at a downstream stage, that is, whether the buffer is full. Hence, we are required to choose a strict synchronization approach.

It is clear that the execution time of a parallel time-driven simulation of a MIN depends on how switches are allocated to each processor. We do not address this problem in

this paper, but rather consider a very simple assignment that is described later in the paper.

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 during the simulations. The experiments and their results are reported in Section 4. Section 5 summarizes the results of our study and describes directions for our future research.

2 Delta Networks

We are interested in a class of MIN's 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 3 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 take 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. 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. In the case of a single buffer system, a packet cannot be transmitted to the next stage if the buffer is full.

Packets arrive to the inputs of the network (first stage) according to some stochastic process. Our studies assume that arrivals to each input are described by a Bernoulli process and that the processes are independent from input to input. Furthermore, we assume that the processes are all identical and that each packet chooses any output with equal probability. We let λ denote the packet arrival rate at each input port, i.e., on each cycle a new packet is generated at each input with probability λ. Last, we are

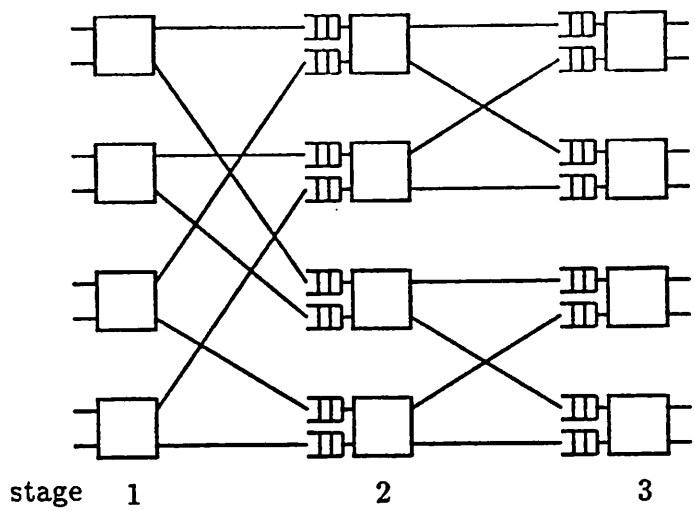

Figure 1: A three stage delta network.

interested in determining the average packet delay and the average buffer occupancies.

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, $1 \leq i \leq 2^{n-1}$, $1 \leq j \leq n$. Stage 1 is the one with the inputs to the MIN and stage n is the one with outputs from the MIN. They are sometime referred to as the first stage and the last stage.

3 The Time-Driven Simulation Approach

In order to understand the different approaches taken to synchronize the processors in our time-driven parallel simulation, it is useful to describe the underlying architecture of the Sequent Symmetry which was used to carry out the experiments. Following the description of the architecture, we will describe ho switches are assigned to processors and the different methods for synchronizing the processors.

3.1 The Sequent Symmetry System

The Sequent Symmetry S81 series system at the University of Massachusetts has 12 Intel 80386 processors and 96 Mbytes of physical memory. All processors are connected to shared memory by a shared bus with a 56.6 Mbytes/sec. transfer rate. Each processor contains 64 Kbytes of cache RAM, which greatly reduces the number of times each processor must access system memory.

Sequent computers run the DYNIX system, a version of UNIXbsd with extensions for processor scheduling. As our simulator never creates more than 8 processes during the experiments, we are able to ensure that each process executes on a different processor.

The Sequent provides specialized hardware to support process synchronization. This consists of a cache of 64 single bit gates and a separate bus for accessing them. These gates are used in a test-and-set manner where a process loops until it acquires a gate. As copies of all 64 gates reside at each processor, the loops do not require use of the bus. These gates can be used to regulate access to locks and counting semaphores which in turn can be used to effect process synchronization. In particular, they support the following higher level primitives that we use in our simulations

spinlock - A lock ensures mutual exclusion to access a shared data structure. To access a shared data structure a process must first acquire the lock associated with the data structure. If the lock is held by another process it spins until the lock is available. This spinning is also called busy waiting. The process releases the lock after it has accessed the shared data structure.

barrier - A 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.

The reader is referred to [9] for further details on these primitives.

3.2 Switch-Processor Assignment and Processor Synchronization

An n stage MIN has 2^{n-1} rows of switches. We allocate switches to processors in the following manner. Let the number of processors, P, be expressible as a power of 2, $P = 2^m$ where $m < n$. If we label these processor $0, 1, \cdots 2^m - 1$, then we assign the switches in rows $j2^{m-n-1} + 1, \cdots, (j+1)2^{m-n-1}$ to the j-th processor.

The simulation is time-driven. During each time unit, a processor simulates the 2^{n-m-1} switches assigned to it at stage n. It removes packets from the network if there are any and collects the statistics. It then simulates the 2^{n-m-1} switches assigned to it at stage $n-1$, selecting and passing packets from stage $n-1$ to stage n. This procedure is repeated for each stage until stage 1 is reached. At this point new packets are generated according to the arrival process and added to the network. The clock then advances by one and the simulation returns to the n-th stage.

In order for the simulation to be correct, it must handle the following constraints,

mutual exclusion - Each buffer is handled by two switches at two adjacent stages. One puts packets into it. The other removes packets from it. Only one of them 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).

In order to ensure that the clocks are synchronized at the processors, we require the processors to wait 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 three approaches for handling synchronization within a single time unit that are required to supplement the barrier. All three approaches are applicable to the simulation of infinite buffer delta networks. Only the first approach is applicable to the simulation of single buffer delta networks.

multibarrier: $n-1$ barriers are used for simulating a network with n stages. All the processors meet at a barrier between simulating switches at two adjacent stages. With such a strict synchronization method it is impossible for two processors to simulate switches at different stages simultaneously. No lock is required for mutual exclusion. Since the simulation moves from the last stage to the first stage, single passage is guaranteed. The computation related to a switch is simple. The basic actions are selecting packets (at most two), moving them out of their buffers, and appending them to buffers at the next stage. If the buffers are finite, the switch has to check if the one to which the packet is destined is full or not.

lock: When the switches have infinite buffers a packet can be moved to the next stage regardless of the buffer occupancy. Therefore, by adding locks to the buffers, the number of barriers used in the multibarrier method can be reduced to one. Locks are used when moving packets out of or into buffers. The processors only need to synchronize at the end of a time unit. To ensure single passage each packet must have a timestamp that indicates when the packet arrives at this buffer. Such timestamps are not needed in the multibarrier approach.

temporary storage: Instead of using locks we can add a pair of variables in front of each input port at each switch in order to ensure mutual exclusion. Each variable is used to track one packet as it moves from one switch to another. The computation required to simulate a switch includes the removal of packets from these new variables and their placement into the input buffers. The packets are then selected and transmitted. The transmission is simulated by placing the packet into one of the variables associated with the next buffer at the next stage. Thus it becomes unnecessary for a processor to access buffers associated with a switch that is allocated to some other processor. The two variables associated with each buffer differ from each other as follows. One variable is used to hold a packet that arrives at odd time units, $t = 2i - 1$, $i = 1, 2, \cdot$, and the other is used to hold packets that arrive during even time units, $t = 2i$, $i = 1, 2, \cdots$. In this approach, processors need only incur a single synchronization delay at the cost of additional computation per switch.

4 Experimental Results

We compared the execution times of the time-driven parallel simulations on 8 processors with the execution times of a serial simulation. The serial simulation was also a time-driven simulation similar to the parallel simulation except that no barrier synchronization, spin locks or additional variables are required.

Figures 2 and 3 illustrate the speedups gained by using the parallel simulator with the multibarrier synchronization approach. Simulations were conducted on both the single and infinite buffer delta network. The number of stages varies from 4 to 9 (corresponding to from 16 to 512 input ports) and the arrival rate, λ, takes values 1/4, 1/2, and 3/4. Because the queues are associated with the input ports, the latter rate is the maximum throughput of the system. Thus as time goes on, the simulation will always find a packet in each queue. We ran each simulation for 20,000 time units in

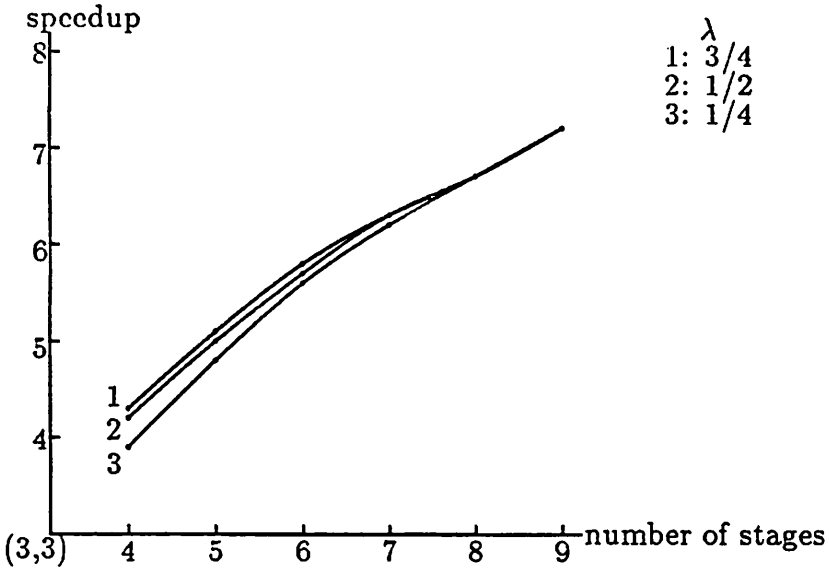

Figure 2: Speedup using multibarrier synchronization for single buffer MIN.

each case. Each simulation was repeated three times (twice for $n = 9$) and the ratio of the average execution times was used to compute speedup.

The potential speedup is 8. As seen in Figures 2 and 3, the real speedup is less than 8. There are a number of reasons why a speedup of 8 was not achieved. First, there are delays and overheads incurred in the parallel simulation due to synchronization. Second, there is a slowdown in the parallel simulation due to the sharing of variables between different processors. This results in a decrease in the cache hit rate, necessitating more accesses to main memory. At this point in time we have been unable to separate out the effects of each of these factors.

We observe that speedup is an increasing function of MIN size. Since the number of processors is fixed, the granularity of computation increases with the MIN size. That is, as the MIN size increases, each processor simulates more switches and therefore does more useful work between synchronizations. In addition, an argument based on the central limit and the maximum of normally distributed random variables indicates that the fraction of time a processor spends idle while it is waiting for synchronization is a decreasing function of MIN size. Let X_i be a random variable that denotes the processing time for processor i between barriers. Assuming the time to perform the barrier synchronization is negligible, the average time betwen barriers is approximately

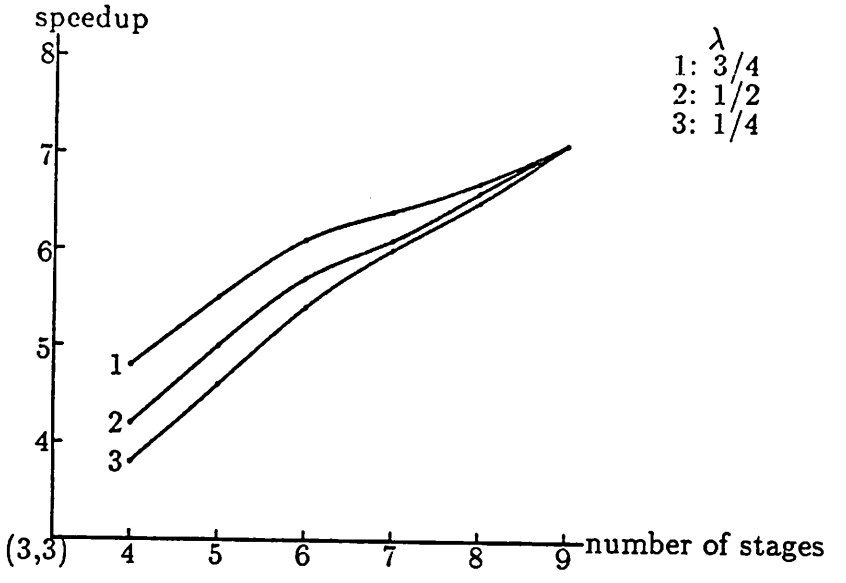

Figure 3: Speedup using multibarrier synchronization for infinite buffer MIN.

$E[\max\{X_i\}]$ and therefore the (relative) increase in computation time due to waiting for synchronization is approximately $E[\max\{X_i\}]/E[X_i]$. Now X_i is the time required to process $k = 2^{n-m-1}$ switches. By the central limit theorem, for large k, X_i is approximately normally distributed with mean $k\mu$ and standard deviation $\sqrt{k}\sigma$ where μ is the average time to process a switch and σ is a standard deviation term. This central limit theorem is valid under quite broad conditions; if the k switch processing times are independent and identically distributed r.v.'s, then σ is the standard deviation of the switch processing time, but if they are dependent r.v.'s, then σ also includes the effect of the autocorrelation [1]. If the X_i's are independent r.v.'s (which they are not), then $E[\max\{X_i\}] \approx k\mu + \sigma\sqrt{2k\ln(m)}$ and therefore $E[\max\{X_i\}]/E[X_i]$ tends to one as the MIN size increases. This argument is valid (under certain circumstances) even if the independence assumption is relaxed [7].

We also observe a slight increase in speedup as λ increases. This is primarily related to the granularity issue as well, since with increasing λ there are more packets in the network for each processor to handle.

Due to a limit on the number of locks one can use in a program on the Sequent we were only able to run parallel simulations based on the lock approach for MINs containing

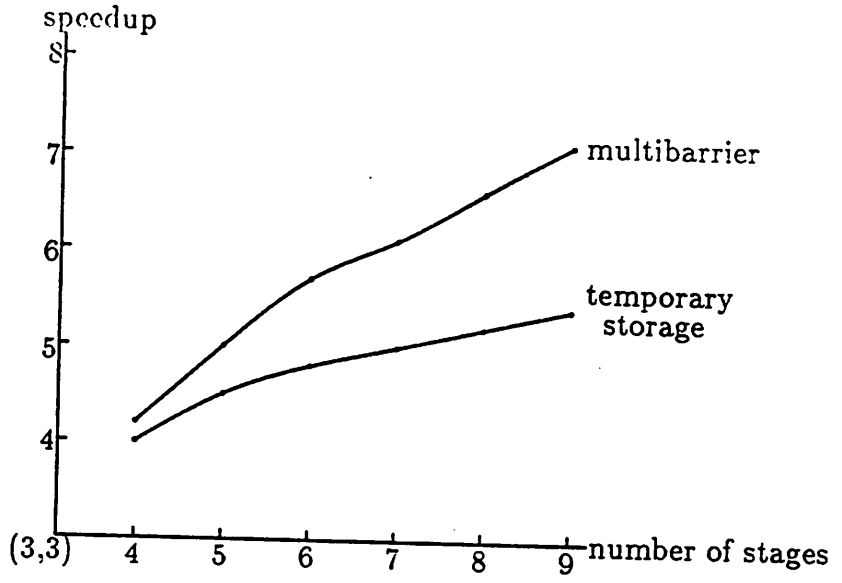

Figure 4: Comparison of the multibarrier and temporary storage synchronization methods for $\lambda = 1/2$.

4 and 5 stages. The performance of this approach is the worst. When $\lambda = 1/2$, the simulation requires 22% more time for a 4 stage MIN, and 30% more time for a five stage network, compared with the simulation using the multibarrier method. The amount of computation needed to process a single switch using the lock method falls between that for the multibarrier method and for the temporary storage method. Thus the degradation is due mainly to the overhead required to lock and unlock shared data structures.

Figure 4 compares the multibarrier method and the temporary storage method. The multibarrier method is superior to the temporary storage method especially when the network is large. This suggests that the tasks are distributed evenly enough among processes so that it is not worthwhile to remove barriers by introducing more computation.

5 Summary and Future Plans

In this paper we reported on a study of time-driven parallel simulation of delta networks. We designed and evaluated several synchronization approaches and found a multibarrier approach to yield the best speedup.

Using the multibarrier approach, it is possible to obtain nearly all of the potential speedup by increasing the problem size. In the future, we will determine whether this observation holds for different number of processors, e.g., 4, 16 processors, and for non-identical arrival processes at the various inputs, e.g., hot spots [12]. In the latter case, we may have to develop different switch-processor assignments in order to obtain good performance.

Even in the case of uniform loads at all input ports, careful processor allocation may improve the performance of the simulation. For example, if we consider a banyan network, it is possible to identify what appear to be natural building blocks consisting of $j2^{j-1}$, $j = 2, 3, \cdots$ switches. No synchronization is required within the block if the whole block is assigned to a single processor. More importantly, this reduction in synchronization can be obtained without increasing the amount of computation required to simulate a switch. Synchronization between these building blocks may be best achieved using the multibarrier method. We are currently investigating this issue.

Last, it will be interesting to compare the approach developed by Nicol [8] with our approach. Nicol reports results for a six stage MIN. However, Nicol's results are not directly comparable with ours since he simulated a heavily loaded closed queueing network model with the queues at the output ports of the switches on 16 processors whereas we simulated a moderately loaded open queueing network with the queues at the input ports on 8 processors. Placement of the queues at the output ports ensures that the network is a FCFS network, however it is not a FCFS Network when the queues are placed at the input ports. Such a comparison will be part of our future work.

References

[1] P. Billingsley. *Convergence of Probability Measures.* Wiley, New York (1968).

[2] L.N. Bhuyan, "Guest Editors Introduction: Interconnection Networks for Parallel

and Distributed Processing", *Computer*, Vol. 20, (June 1987), 9-12.

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

[4] D.M. Dias and J.R. Jump, "Packet Switching Interconnection Networks for Modular Systems", *Computer*, Vol.14, (Dec. 1981), 43-54.

[5] V. Holmes, *Parallel Algorithms on Multiple Processor Architectures*, Ph.D. Thesis, Dept. Comp. Sci., Univ. Texas, Austin, Texas, (1978).

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

[7] M.R. Leadbetter, G. Lindgren, and H. Rootzen. *Extremes and Related Properties of Random Sequences and Processes.* Springer Verlag, New York (1983).

[8] D.M. Nicol, "Parallel Discrete-Event Simulation of FCFS Stochastic Queueing Networks," ICASE Report No. 88-29, Institute for Computer Application in Science and Engineering, Hampton, Virginia (May 1988). To appear in the *Proceedings of the ACM SIGPLAN Symposium on Parallel Programming, Environments, Applications, and Languages.* Yale University, (July 1988).

[9] Osterhaug, A., *Guide to Parallel Programming on Sequent Computer Systems*, Sequent Computer Systems, Inc., (1985).

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

[11] G.F. Pfister, et al., "The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture," *Proc. Intntl. Conf. Parallel Processing*, (1985), 764-771.

[12] 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.

[13] G.F. Pfister, "The IBM Yorktown Simulation Engine," *Proc. of the IEEE*, Vol. 74, No. 6, (June 1986), 850-860.

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

[15] R. Thomas, "Behavior of the Butterfly Parallel Processor in the Presence of Memory Hot Spots," *Proc. Intntl. Conf Parallel Processing*, (1986), 46-50.

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