Proceedings of the 1993 Winter Simulation Conference
G.W. Evans, M. Mollaghasemi, E.C. Russell, W.E. Biles (eds.)

PARALLEL AND DISTRIBUTED DISCRETE EVENT SIMULATION: ALGORITHMS AND APPLICATIONS

Richard M. Fujimoto

College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280, U.S.A.

ABSTRACT

This tutorial reviews issues concerning the execution of discrete event simulation programs on multiprocessor and distributed computing platforms. The synchronization problem that has driven much of the research in this field is reviewed. Space-parallel and time-parallel approaches to concurrent execution are described. Experiences in applying these techniques to specific applications are examined. Finally, issues that must be considered to develop efficient concurrent simulation programs are discussed.

1 INTRODUCTION

Concurrent discrete event simulation (or simply concurrent simulation) refers to the execution of a discrete event simulation program on a machine containing multiple CPUs. This includes execution on closely coupled multiprocessor machines (*parallel simulation*) as well as geographically distributed computers (*distributed simulation*). The field has been studied over the last 15 years.

Concurrent simulation is of practical importance because many simulations require excessive amounts of CPU time. For example, simulations of only modest sized telecommunication networks may require weeks on a modern workstation [Roberts, 1992]. Parallel and distributed computers are becoming the most powerful machines that can be assembled, making exploitation of this technology very attractive. Some simulations require geographically distributed resources, e.g., humans, or databases that are costly and/or difficult to move. These factors have fueled the growing interest in concurrent simulation techniques.

What is a concurrent simulator? A discrete event simulation computes a sample path through the set of possible system states for a period of simulated time. A concurrent simulator attempts to construct this sample path by utilizing many processors. Two approaches have been employed to accomplish this task. The *space-parallel* approach partitions the system being modeled into a collection of subsystems, and assigns a *logical process (LP)* to simulate each one. Different LPs may execute concurrently on different processors, and communicate exclusively by exchanging timestamped *events* or *messages*. Each LP is essentially a sequential, event-driven simulator that computes the portion of the sample path pertaining to the subsystem it models. Sending an event to a logical process is equivalent to scheduling that event in the receiving processor's list of pending events. A key question, referred to as the *synchronization problem*, is ensuring that the *events* are processed in a correct order.

An alternative called the *time-parallel* approach partitions the simulated time axis into intervals $[T_1, T_2]$, $[T_2, T_3]$, ..., $[T_i, T_{i+1}]$, ... and assigns each to a separate processor. For instance, processor i might compute the sample path for $[T_i, T_{i+1}]$. The final state in the sample path for $[T_{i-1}, T_i]$ must match the initial state in the sample path for $[T_i, T_{i+1}]$. Thus, this approach relies on being able to perform the simulation corresponding to the *ith* interval without first completing the simulations of the preceding $(i-1, i-2, ...1)$ intervals. This is typically accomplished by a parallel computation that rapidly computes the states in question (e.g., using parallel prefix [Greenberg *et al.*, 1991]) or relaxation [Lin and Lazowska, 1991].

Time-parallel simulations offer the potential for massive parallelism because executions usually span long periods of simulated time. However, they are currently not as robust as space-parallel approaches because they rely on specific properties of the system being modeled, e.g., specification of the system's behavior as recurrence equations and/or a relatively simple state descriptor. This approach is currently limited to a handful of applications, e.g., queue-

ing and petri networks, cache memories, and statistical multiplexors in telecommunication networks. Space-parallel simulations offer greater robustness and wider applicability, but concurrency is limited to the number of logical processes. Sometimes both time and space parallelism can be used [Gaujal *et al.*, 1993].

Surveys of concurrent simulation techniques have appeared [Misra, 1986, Richter and Walrand, 1989, Fujimoto, 1990a]. Recently, Fujimoto and Nicol [Fujimoto and Nicol, 1992] surveyed developments concerning synchronization, performance analysis, time parallelism, hardware support, load balancing, and memory management. Here, a major emphasis is on applications and considerations that modelers must make to achieve good performance.

2 SYNCHRONIZATION ALGORITHMS

The synchronization problem in space-parallel simulations has spawned an enormous amount of research. Here, *synchronization* is concerned with ensuring that events are processed in an order that yields the same results as the sequential execution, where events are processed in timestamp order. To achieve this, the concurrent execution must process dependent events in timestamp order. Because all interactions between LPs occur through event messages, it is sufficient that *each* LP process events in timestamp order.

Two classes of synchronization algorithms have been proposed. So-called *conservative* algorithms guarantee that dependent events are never processed out of order, i.e., synchronization errors never occur. *Optimistic* algorithms allow synchronization errors to occur, but provide a mechanism to recover. A more detailed taxonomy is described in [Reynolds, Jr., 1988], but this simple dichotomy will suffice here.

2.1 Conservative Algorithms

Conservative mechanisms must determine when it is "safe" to process an event, i.e., when have all events on which this event depends been processed? To realize this, an LP cannot process an event with timestamp T until it can guarantee that no event with timestamp smaller than T will later arrive.

Chandy and Misra [Chandy and Misra, 1979], and Bryant [Bryant, 1977] developed some of the first algorithms. They assume the topology indicating which LPs send messages to which others is known, and messages arrive on each incoming link in timestamp order. This guarantees that the timestamp of the last message received on a link is a lower bound on the timestamp of any subsequent message arriving on that link. This information allows the LP to determine a lower bound on the timestamp of future messages, allowing it to determine if an event is safe to process. If an LP has no safe events, it must block. This blocking can lead to deadlock.

Null messages are used to avoid deadlock. A null message with timestamp T_{null} sent from LP_A to LP_B is a promise by LP_A that it will not send a message to LP_B carrying a timestamp smaller than T_{null}. Null messages do not correspond to any activity in the simulated system. Processes send null messages on each outgoing link after processing each event. A null message provides the receiver with additional information that may be used to determine that other events are safe to process. It can be shown that this algorithm avoids deadlock under some mild constraints [Chandy and Misra, 1979].

This algorithm may generate many null messages, however. Chandy and Misra [Chandy and Misra, 1981] later developed another approach that allows the computation to deadlock, but then detects and breaks it.

Numerous other conservative approaches have since been proposed. Space does not permit elaboration of them here, but they are reviewed in [Fujimoto, 1990a, Fujimoto and Nicol, 1992]. One technique does require mentioning as it is referred to later. Some protocols use a synchronous execution where the computation cycles between (i) determining which events are "safe" to process, and (ii) processing those events [Ayani, 1989, Chandy and Sherman, 1989, Lubachevsky, 1989, Nicol, 1993]. To determine which events are safe, the *distance between LPs* is sometimes used. This "distance" is the minimum amount of simulation time that must elapse for an event in one LP to directly or indirectly affect another LP [Ayani, 1989, Lubachevsky, 1989], and can be used by an LP to determine bounds on the timestamp of future events it might receive from other LPs.

2.2 Optimistic Algorithms

The Time Warp mechanism invented by Jefferson and Sowizral [Jefferson and Sowizral, 1982, Jefferson, 1985] is the most well known optimistic method. When an LP receives an event with timestamp smaller than one or more events it has already processed, it rolls back and reprocesses those events in timestamp order. Rolling back an event involves restoring the state of the LP to that which existed prior to processing the event (checkpoints are taken for this purpose), and "unsending" messages sent by the rolled back events. An elegant mechanism called anti-messages is provided to "unsend" messages.

An anti-message is a duplicate copy of a previously sent message. Whenever an anti-message and its matching (positive) message are both stored in the same queue, the two are deleted (annihilated). To "unsend" a message, a process need only send the corresponding anti-message. If the matching positive message has already been processed, the receiver process is rolled back, possibly producing additional anti-messages. Using this recursive procedure all effects of the erroneous message will eventually be erased.

A number of other optimistic algorithms have been proposed [Fujimoto, 1990a, Fujimoto and Nicol, 1992]. Most attempt to limit the amount of optimistic computation. Rather than focusing on these approaches, we instead turn our attention to applications and experiences using these techniques.

3 APPLICATIONS

It is convenient to characterize applications as either *static* or *dynamic*. Static applications utilize a fixed topology of LPs. Dynamic applications vary the communication pattern among LPs, and possibly the number of LPs during the simulation.

3.1 Static Applications

Static applications are often simulations of a network of components, e.g., digital circuits or switching nodes in a telecommunication network. The null message and deadlock detection and recovery algorithms as well as some others that rely on "distance between LPs" require a static network topology.

Queueing Networks and Synthetic Workloads. Queueing networks are a standard benchmark for evaluating concurrent simulation protocols. They represent a "stress test" because the amount of computation per event is small, so overheads in the synchronization mechanism become readily apparent.

The literature is filled with queueing network simulations that achieve good performance for both conservative and optimistic synchronization protocols. Reed et al. [Reed *et al.*, 1988] describe perhaps the first empirical measurements of performance of the null message and deadlock detection and recovery algorithms on a Sequent multiprocessor, but report disappointing performance. Fujimoto [Fujimoto, 1989a] reports much better performance for these applications for first-come-first-serve queues when the program was written to exploit lookahead. Fujimoto [Fujimoto, 1989b] also reports speedups as high as 57 using 64 processors on a BBN Butterfly using Time Warp, and reports that Time Warp outperforms these conservative algorithms for queues with preemption. Others report good speedup using different concurrent simulation algorithms, e.g., see [Chandy and Sherman, 1989, Ayani, 1989, Nicol and Heidelberger, 1993], among others.

A growing number of algorithms use time-parallel simulation techniques to achieve massively parallel simulations of queues. Greenberg et al. [Greenberg *et al.*, 1991] formulate the behavior of a G/G/1 queue as linear recurrence equations, and then solve these equations using parallel prefix. Their approach extends to certain types of feed-forward networks. Related approaches include [Lin, 1993, Lin and Lazowska, 1991, Wang and Abrams, 1992, Ammar and Deng, 1992, Nikolaidis *et al.*, 1992].

Fujimoto [Fujimoto, 1990b] describes an extension to the "hold" model that has been used to evaluate priority queue implementations in sequential simulations. This parallel hold (PHOLD) model characterizes symmetric workloads, and has been used for empirical and analytic studies of Time Warp [Gupta *et al.*, 1991, Fujimoto, 1990b] and conservative algorithm [Fujimoto, 1989a] performance. An asymmetric version has also been proposed [Gupta, 1993].

Digital Logic Circuits. Simulation of logic circuits is of considerable interest to the electronic computer-aided-design community because they are a major bottleneck in the design cycle. LPs model components such as memories, arithmetic circuits, or individual gates.

Su and Seitz [Su and Seitz, 1989] use variations of the null message algorithm to speed up gate-level simulations on an Intel iPSC computer. Although speedups are relatively modest (8 using 64 processors, and 10 to 20 using 128 processors), they conjecture that better performance could be obtained on shared-memory machines where the overhead of sending null messages is reduced. However, Soulé and Gupta [Soulé and Gupta, 1991] examine gate-level simulations using deadlock detection and recovery on shared-memory machines, and report disappointing results. Although they achieve speedup relative to a sequential execution of the concurrent program ranging from 7 to 9, they go on to report that the simulator is actually *slower* than a traditional parallel logic simulation that uses a global clock and only allows concurrent execution of events containing the same timestamp. The Soulé and Gupta study examines realistic circuits ranging from a multiplier to an entire CPU.

On the other hand, Bailey [Bailey, 1989] points out that the timing model plays a critical role in evaluating the effectiveness of asynchronous simulation algo-

rithms relative to traditional parallel logic simulators using a global clock. She indicates that the global clock approach offers little hope of speedup if propagation delays are variable from one gate to the next (variable-delay timing models). Soulé and Gupta use the so-called "unit delay" model where each gate has a delay of one unit of time, so Bailey's conclusion is consistent with their results.

Nandy and Loucks [Nandy and Loucks, 1992] and Davoren [Davoren, 1989] study partitioning and mapping algorithms used with the null message algorithm. DeBenedictis et al. [DeBenedictis *et al.*, 1991] develop a conservative simulation algorithm that transforms cyclic networks into acyclic ones, and demonstrate the algorithm on two cross-coupled NAND gates.

Using Time Warp, Briner [Briner, 1991] reports speedups ranging from 10 to 25 for simulations of a variety of gate- and transistor-level circuits. Sporrer and Bauer [Sporrer and Bauer, 1993] also report speedups ranging from 4 to 8 in using Time Warp for a variety of circuits, with speedup saturating at approximately 12 processors.

A few studies compare the performance of different synchronization algorithms for concurrent logic simulation. Lin et al. [Lin *et al.*, 1990] argues that Time Warp offers better performance than conservative algorithms based on a critical path analysis. Chung and Chung [Chung and Chung, 1991] compare the null message algorithm, Time Warp, Time Warp augmented with time windows [Sokol *et al.*, 1988], and the global clock algorithm, and indicate Time Warp with time windows yielded the best performance on an SIMD Connection Machine. Manjikian and Loucks [Manjikian and Loucks, 1993] report that optimistic synchronization algorithms perform somewhat better than conservative ones for logic simulations performed on networked workstations.

Computer Architectures. Logical processes in an architecture-level simulation model CPUs or switches rather than individual gates. Concurrent simulation has been applied to simulating critical components as well as the entire parallel computer.

Yu et al. [Yu *et al.*, 1989] and Goli et al. [Goli *et al.*, 1990] study the use of a synchronous timestepped approach to simulate multistage interconnection networks (MINs), and report speedups of over 14 on a 16 processor Sequent Symmetry. Ayani and Rajaei [Ayani and Rajaei, 1990] also demonstrate good performance using a conservative algorithm based on the distance between objects.

Concurrent simulation of cache memories driven by memory reference traces have also been studied. Lin et al. [Lin *et al.*, 1989] propose a technique that features minimal communication and linear speedup. In one of the first approaches to utilize *time-parallel* simulation, Heidelberger and Stone [Heidelberger and Stone, 1990] simulate uniprocessor caches using a "fix-up" procedure to perform state-matching. Nicol, et al. [Nicol *et al.*, 1992] also employ time-parallel simulation techniques to uniprocessor cache simulations, and demonstrate their algorithm on a Masspar SIMD machine.

The Wisconsin Wind Tunnel project [Reinhardt *et al.*, 1993] reports good performance in simulating a shared memory multiprocessor using so-called direct execution techniques that execute application programs directly on the simulator host, rather than through an interpreter. A synchronous conservative algorithm is used where all instructions within a time window can be executed concurrently because the size of the window is set to be no larger than the minimum distance between any two LPs in the simulation. Konas and Yew [Konas and Yew, 1991, Konas and Yew, 1992] compare the performance of the null message algorithm with Time Warp and a synchronous approach using a global clock for simulating a synchronous multiprocessor system and an asynchronous network. Bailey et al. [Bailey and Pagels, 1991, Bailey *et al.*, 1993] study parallel simulation of parallel architectures using point-to-point connections and bus-based multiprocessors using the null message algorithm, and report encouraging results.

Telecommunication Networks. The increasing use of fiber optics technology and the need to integrate different types of traffic (voice, video, data, fax) in so-called broadband integrated services digital networks (B-ISDN) have heightened the need for high performance simulation techniques to aid in network design. Several have obtained significant speedup. Earnshaw and Hind [Earnshaw and Hind, 1992] report up to an order of magnitude speedup in simulating B-ISDN networks.

Gaujal et al. [Gaujal *et al.*, 1993] report an order of magnitude speedup in simulating call routing using time and space-parallel techniques, and indicate that simulator performance exceeds 3,000,000 calls per minute. Nikolaidis et al. [Nikolaidis *et al.*, 1992] describe a time-parallel algorithm that exploits the bursty structure of telecommunication traffic in simulating ATM (asynchronous transfer mode) multiplexors. They report performance between 10^6 and 10^9 simulated cell arrivals per second of real time using 30 processors of a KSR-1 machine. Turner and Xu [Turner and Xu, 1992] report success in speeding up telephone switching network simulations using a vari-

ation of Time Warp. Lomow [Lomow, 1992] reports an order of magnitude speedup was obtained in simulating telephone switching networks on Time Warp. Mouftah and Sturgeon [Mouftah and Sturgeon, 1990] report that a distributed simulation executing on two workstations provides a modest speedup (up to 20%) for certain test cases.

Not all report success, however. Phillips and Cuthbert [Phillips and Cuthbert, 1991] report that their Transputer-based conservative simulations ran more slowly than a sequential simulator. Tallieu and Verboven [Tallieu and Verboven, 1991] also report disappointing performance in simulating an Ethernet.

Transportation, Manufacturing, and Others. Use of concurrent simulation techniques in a variety of other applications have also been studied. Merrifield et al. [Merrifield *et al.*, 1990] studied the use of the null message algorithm in simulating a road transportation network. Traffic for 292 vehicles and 292 road junctions were simulated in this study. Approximately an order of magnitude speedup was obtained.

Nevison [Nevison, 1990] examines the use of concurrent simulation in manufacturing applications. In particular, he develops a conservative synchronization algorithm that exploits the structure of the application, namely collections of closed loops that frequently occur in flexible manufacturing systems.

Baezner et al. [Baezner *et al.*, 1993] simulate a health care system consisting of a hierarchical network of hospitals. Hospitals at higher levels in the hierarchy have a higher degree of capability. Patients move up the hierarchy if their illness cannot be handled by the local facility. They report 18 fold speedup using Time Warp executing on 32 processors of a Transputer network. A 20 millisecond delay was added to each event to reduce the effect of overhead computations, and expose the amount of parallelism that the simulator is able to exploit.

Lubachevsky [Lubachevsky, 1989] applies conservative synchronization techniques to simulate Ising spin. Atoms are assigned to positions in a two dimensional grid, and spin in one of two directions. They attempt to change the direction of their spin at randomly selected points in time. At each attempt, a new spin direction is computed based on the direction of spin of neighboring atoms as well as the spin value of the atom attempting to change. Speedups as high as 1900 are reported on a 16,384 processor Connection Machine (CM-1).

Other researchers have applied concurrent simulation techniques to simulate timed Petri networks. Thomas and Zahorjan [Thomas and Zahorjan, 1991] obtain an order of magnitude speedup using a conservative protocol tailored to this application. Kumar and Harous [Kumar and Harous, 1990] describe a variation on the null message algorithm for this purpose. Nicol and Roy [Nicol and Roy, 1991] simulate timed Petri nets using a conservative windowing protocol. Baccelli and Canales [Baccelli and Canales, 1993] use time-parallel techniques to achieve massively parallel simulations.

3.2 Dynamic Applications

Dynamic applications often involve a number of physical entities moving over a terrain, and interacting, e.g., based on physical proximity to other entities. Applications include combat models, biological systems, and cellular telephone networks.

A typical operation in such a simulation is an entity, e.g., a combat unit, will scan other entities in its immediate vicinity, and then act according to some plan, e.g., attack or retreat. In order to determine physical proximity the map is typically partitioned into a checkerboard-like grid. This means that each unit need only check a few neighboring grid sectors to determine which units are close by rather than examine the location of all units in the system. The grid structure typically uses square or hexagonal shaped grids.

It is important to realize that most concurrent synchronization algorithms (e.g., null messages, Time Warp, etc.) do not allow LPs to access common state variables, i.e., no *shared state* is allowed. Thus, a typical approach is to use a logical process to implement each grid sector. The intercommunication pattern is dynamic because the LPs modeling the moving entities will communicate with different grid sector LPs depending on their current location.

Sharks, Ants, and Colliding Pucks. Several benchmark applications have been proposed for dynamic applications. "Shark's world" [Conklin *et al.*, 1990] consists of an ocean and a set of migrating sharks and fish. The sharks swim in a random pattern until they come within a certain distance of a fish. It then attacks the fish, and devours it. This benchmark was implemented by Presley et al. [Presley *et al.*, 1990] using Time Warp, Bagrodia and Liao [Bagrodia and Liao, 1990] using another optimistic algorithm, and Nicol [Nicol and Riffe, 1990] using a conservative scheme. All three were able to achieve speedup, although the conservative program does so by precomputing the sharks' path, which may not always be possible.

Two other related benchmarks are the colliding pucks [Hontalas *et al.*, 1989] and the ant foraging

[Ebling *et al.*, 1989] models developed at JPL. In the first, a set of pucks slide over a frictionless surface and deflections are simulated as the pucks collide with each other or the edge of the surface. "Ant world" is in a similar spirit as shark's world, and consists of ants moving over a terrain foraging for food. Logical processes are used to model ants, nests, and grid sectors that define the terrain. Both "pucks" and "ant world" demonstrated good speedup using Time Warp.

Combat Models. Perhaps the most substantive dynamic application to which concurrent simulation techniques have been applied thus far are combat models. Researchers at JPL [Wieland *et al.*, 1989] describe a combat model implemented on Time Warp. The simulation consists of two opposing armies moving through three major phases: an advance phase where the armies approach each other, a conflict phase where they fight, and a clean up phase where the outcome of the battle has been determined, but minor pockets of fighting remain. Thirty-fold speedup on a BBN Butterfly is reported for this application.

Morse [Morse, 1990] reports success in speeding up a combat model developed in Modsim, an object-oriented simulation language, running on top of the JPL Time Warp Operating System (TWOS). On the other hand, Rich and Michelsen [Rich and Michelsen, 1991] report a failed effort to port a combat model to the Modsim/TWOS system. A major problem was the model contained many thousands of very small objects, while TWOS was designed to support perhaps a few hundred large-grained objects. Sufficient "hooks" were not available in the compiler to properly map the application to the TWOS software.

Steinman [Steinman, 1992] examines the question of determining which aircraft/radar can "see" each other (proximity detection) in optimistic simulations and demonstrates linear speedup up to 32 processors on a hypercube machine. A convincing demonstration of distributed simulation technology is the Simnet project [Kanarick, 1991] for training military personnel, although the synchronization techniques used there differ substantially from the methods discussed here.

4 DEVELOPING EFFICIENT CODE

At present, achieving speedup requires expertise in both simulation modeling and parallel computation. In principal, synchronization algorithms such as Time Warp free the programmer from specifying program synchronization, however, poor performance will result if proper precautions are not taken. A few important issues are discussed next.

Lookahead. Performance of both conservative and optimistic synchronization algorithms is usually improved if events are scheduled far into the simulated future. This allows events with low timestamps to be processed concurrently without being affected (e.g., by rollback in Time Warp or blocking in conservative protocols) by events that will be produced later. Failure to exploit lookahead can result in a dramatic performance degradation, particularly for conservative protocols [Fujimoto, 1990a].

Granularity. All synchronization algorithms entail a certain amount of overhead computation in processing each event. An LP must be scheduled and executed, data structures manipulated, messages sent, etc. In Time Warp, additional overheads for state saving and rollback are also required. If the amount of computation per event is very small, most of the simulator's time will be spent in the overhead computations, degrading performance. Increasing event granularity may be difficult, however, depending on the application.

Push vs. Pull Processing. In many applications, particularly dynamic ones, LPs must access state that is stored in another LP, e.g., information in grid sector LPs. A simple approach is to send a message to the grid sector to obtain this information as it is needed [Wieland and Jefferson, 1989]. The drawback of this "pull processing" approach is that the LP must wait until the information is delivered. This may be very time consuming.

A more efficient approach is to transmit information to processes in advance of when it is needed. This "push processing" approach has been demonstrated to overcome the performance problems described above [Wieland and Jefferson, 1989]. It does, however, greatly complicate the coding of the simulation, and thereby inflate development time and detract from the software's maintainability. Because multiple copies of modifiable data may be distributed across several processes, the application program must be careful to maintain consistency among the multiple copies.

Self-Driven Processes. Consider a source process in a queueing network simulation that generates jobs for the network. This might be implemented by an LP that continually sends messages to itself to advance

in simulated time, and generates new "job" messages as needed. Such "self-driven" processes never receive messages from other LPs, so they may not roll-back or block. Instead, they may flood the system with messages, straining the synchronization protocol and memory management system. If no flow control mechanism is provided, the program may not be able to run at all.

One solution is to program the simulation without self-driven processes. For instance, a protocol might be established where processes receiving new jobs will periodically ask for additional job arrivals rather than allowing the source to generate an excessive number of jobs.

5 CONCLUSIONS

Although many successes have been reported in applying concurrent simulation techniques, this technology has *not* been widely embraced by the general simulation community. Important reasons for this lack of penetration include the current high level of expertise required to effectively exploit the technology, and a lack of development tools and concurrent simulation products. Migrating large, existing sequential simulation models to concurrent environments is difficult. To fully realize its potential, much additional work is required to make concurrent simulation technology more accessible to the general simulation community.

REFERENCES

Ammar, H., and S. Deng. 1992. Time warp simulation using time scale decomposition. *ACM Transactions on Modeling and Computer Simulation*, 2(2):158–177, April.

Ayani, R., and H. Rajaei, 1990. Parallel simulation of a generalized cube multistage interconnection network. In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 22, pages 60–63. SCS Simulation Series, January.

Ayani, R. 1989. A parallel simulation scheme based on the distance between objects. In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 21, pages 113–118. SCS Simulation Series, March.

Baccelli, F. and M. Canales. 1993. Parallel simulation of stochastic petri nets using recurrences equations. *ACM Transactions on Modeling and Computer Simulation*, 3(1):20–41, January.

Baezner, D., G. Lomow, and B. Unger. 1993. A parallel simulation environment based on Time Warp. *International Journal of Computer Simulation*, to appear.

Bagrodia, R. L. and W-T. Liao. 1990. Parallel simulation of the shark's world problem. In *Proceedings of the 1990 Winter Simulation Conference*, pages 191–198, New Orleans, December.

Bailey, M. L. and M. A. Pagels. 1991. Measuring the overhead in conservative parallel simulations of multicomputer programs. In *Proceedings of the 1991 Winter Simulation Conference*, pages 627–636, Phoenix, December.

Bailey, M. L., M. A. Pagels, and K. K. Wong. 1993. How using busses in multicomputer programs affects conservative parallel simulation. In 7^{th} *Workshop on Parallel and Distributed Simulation*, volume 23, pages 93–100. SCS Simulation Series, May.

Bailey, M. L. 1989. The on-chip parallelism of VLSI circuits. Technical Report 89-08-05, University of Washington, Seattle, July.

Briner Jr., J. 1991. Fast parallel simulation of digital systems. In *Advances in Parallel and Distributed Simulation*, volume 23, pages 71–77. SCS Simulation Series, January.

Bryant, R. E. 1977. Simulation of packet communication architecture computer systems. MIT-LCS-TR-188, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Chandy, K. M. and J. Misra. 1979. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering*, SE-5(5):440–452, September.

Chandy, K. M. and J. Misra. 1981. Asynchronous distributed simulation via a sequence of parallel computations. *Communications of the ACM*, 24(4):198–205, April.

Chandy, K. M. and R. Sherman. 1989. The conditional event approach to distributed simulation. In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 21, pages 93–99. SCS Simulation Series, March.

Chung, M. and Y. Chung. 1991. An experimental analysis of simulation clock advancement in parallel logic simulation on an SIMD machine. In *Advances in Parallel and Distributed Simulation*, volume 23, pages 125–132. SCS Simulation Series, January.

Conklin, D., J. Cleary, and B. Unger. 1990. The sharks world (a study in distributed simulation design. In *Distributed Simulation*, volume 22, pages 157–160. SCS Simulation Series, Jan.

Davoren, M. 1989. A structural mapping for parallel digital logic simulation. In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 21, pages 179–182. SCS Simulation Series, March.

DeBenedictis, E., S. Ghosh, and M.-L. Yu. 1991. A novel algorithm for discrete-event simulation. *Computer*, 24(6):21–33, June.

Earnshaw, R. W. and A. Hind. 1992. A parallel simulator for performance modelling of broadband telecommunication networks. In *1992 Winter Simulation Conference Proceedings*, pages 1365–1373, December.

Ebling, M., M. DiLorento, M. Presley, F. Wieland, and D. R. Jefferson. 1989. An ant foraging model implemented on the Time Warp Operating System. In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 21, pages 21–26. SCS Simulation

Series, March.

Fujimoto, R. M. and D. M. Nicol. 1992. State of the art in parallel simulation. In *1992 Winter Simulation Conference Proceedings*, pages 122–127, December.

Fujimoto, R. M. 1989a. Performance measurements of distributed simulation strategies. *Transactions of the Society for Computer Simulation*, 6(2):89–132, April.

Fujimoto, R. M. 1989b. Time Warp on a shared memory multiprocessor. *Transactions of the Society for Computer Simulation*, 6(3):211–239, July.

Fujimoto, R. M. 1990a. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, October.

Fujimoto, R. M. 1990b. Performance of Time Warp under synthetic workloads. In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 22, pages 23–28. SCS Simulation Series, January.

Gaujal, B., A. G. Greenberg, and D. M. Nicol. 1993. A sweep algorithm for massively parallel simulation of circuit-switched networks. *Journal of Parallel and Distributed Computing*, August. To appear.

Goli, P., P. Heidelberger, D. Towsley, and Q. Yu. 1990. Processor assignment and synchronization in parallel simulation of multistage interconnection networks. In *Distributed Simulation*, volume 22, pages 181–187. SCS Simulation Series, January.

Greenberg, A. G., B. D. Lubachevsky, and I. Mitrani. 1991. Algorithms for unboundedly parallel simulations. *ACM Transactions on Computer Systems*, 9(3):201–221, August.

Gupta, A., I. F. Akyildiz, and R. M. Fujimoto. 1991. Performance analysis of Time Warp with multiple homogeneous processors. *IEEE Transactions on Software Engineering*, 17(10):1013–1027, October.

Gupta, A. 1993. Performance analysis of Time Warp mechanism for parallel discrete event simulation. Technical report, Georgia Institute of Technology, Seattle, Georgia, May.

Heidelberger, P. and H. Stone. 1990. Parallel trace-driven cache simulation by time partitioning. In *Proceedings of the 1990 Winter Simulation Conference*, pages 734–737, New Orleans, December.

Hontalas, P., B. Beckman, M. DiLorento, L. Blume, P. Reiher, K. Sturdevant, L. Van Warren, J. Wedel, F. Wieland, and D. R. Jefferson. 1989. Performance of the colliding pucks simulation on the Time Warp Operating System. In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 21, pages 3–7. SCS Simulation Series, March.

Jefferson, D. R. and H. Sowizral. 1982. Fast concurrent simulation using the Time Warp mechanism, part I: Local control. Technical Report N-1906-AF, RAND Corporation, December.

Jefferson, D. R. 1985. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, July.

Kanarick, C. 1991. A technical overview and history of the simnet project. In *Advances in Parallel and Distributed Simulation*, volume 23, pages 104–111. SCS Simulation Series, January.

Konas, P. and P.-C. Yew. 1991. Parallel discrete event simulation on shared-memory multiprocessors. In *Proceedings of the 24th Annual Simulation Symposium*, volume 21, pages 134–148. IEEE Computer Society Press, April.

Konas, P. and P.-C. Yew. 1992. Synchronous parallel discrete event simulation on shared-memory multiprocessors. In 6^{th} *Workshop on Parallel and Distributed Simulation*, volume 24, pages 12–21. SCS Simulation Series, January.

Kumar, P. and S. Harous. 1990. An approach towards distributed simulation of timed petri nets. In *Proceedings of the 1990 Winter Simulation Conference*, pages 428–435, New Orleans, LA., December.

Lin, Y.-B. and E.D. Lazowska. 1991. A time-division algorithm for parallel simulation. *ACM Transactions on Modeling and Computer Simulation*, 1(1):73–83, January.

Lin, Y-B., J-L. Baer, and E. D. Lazowska. 1989. Tailoring a parallel trace-driven simulation technique to specific multiprocessor cache coherence protocols. In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 21, pages 185–190. SCS Simulation Series, March.

Lin, Y-B., E. D. Lazowska, and M. L. Bailey. 1990. Comparing synchronization protocols for parallel logic level simulation. In *Proceedings of the 1990 International Conference on Parallel Processing*, volume 3, pages 223–227, August.

Lin, Y-B. 1993. Parallel trace-driven simulation for packet loss in finite-buffered voice multiplexors. *Parallel Computing*, to appear.

Lomow, G. 1992. private communication, December.

Lubachevsky, B. D. 1989. Efficient distributed event-driven simulations of multiple-loop networks. *Communications of the ACM*, 32(1):111–123, January.

Manjikian, N. and W. M. Loucks. 1993. High performance parallel logic simulation on a network of workstations. In 7^{th} *Workshop on Parallel and Distributed Simulation*, volume 23, pages 76–84. SCS Simulation Series, May.

Merrifield, B. C., S. B. Richardson, and J. B. G. Roberts. 1990. Quantitative studies of discrete event simulation modelling of road traffic. In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 22, pages 188–193. SCS Simulation Series, January.

Misra, J. 1986. Distributed-discrete event simulation. *ACM Computing Surveys*, 18(1):39–65, March.

Morse, K. 1990. Parallel distributed simulation in Modsim. In *Proceedings of the 1990 International Conference on Parallel Processing*, volume 3, pages 210–217, August.

Mouftah, H. T. and R. P. Sturgeon. 1990. Distributed discrete event simulation for communication networks. *IEEE Journal on Selected Areas in Communications*, 8(9):1723–1734, December.

Nandy, B. and W. Loucks. 1992. An algorithm for partitioning and mapping conservative parallel simulation onto multicomputers. In 6^{th} *Workshop on Parallel and*

Distributed Simulation, volume 24, pages 139–146. SCS Simulation Series, Jan.

Nevison C. 1990. Parallel simulation of manufacturing systems: Structural factors. In *Distributed Simulation*, volume 22, pages 17–22. SCS Simulation Series, January.

Nicol, D. M. and S. Roy. 1991. Parallel simulation of timed petri nets. In *Proceedings of the 1991 Winter Simulation Conference*, pages 574–583, Phoenix, Arizona, December 1991.

Nicol, D. M. and P. Heidelberger. 1993. Parallel algorithms for simulating continuous time markov chains. In 7^{th} *Workshop on Parallel and Distributed Simulation*, volume 23, pages 11–18. SCS Simulation Series, May.

Nicol, D. M. and S. E. Riffe. 1990. A conservative approach to parallelizing the shark's world simulation. In *Proceedings of the 1990 Winter Simulation Conference*, pages 186–190, New Orleans, December.

Nicol, D. M., A. Greenberg, B. Lubachevsky, and S. Roy. 1992. Massively parallel algorithms for trace-driven cache simulation. In 6^{th} *Workshop on Parallel and Distributed Simulation*, volume 24, pages 3–11. SCS Simulation Series, January.

Nicol, D.M. 1993. The cost of conservative synchronization in parallel discrete-event simulations. *Journal of the ACM*, April.

Nikolaidis, I., R. M. Fujimoto, A. Cooper, T. Ott, and T. V. Lakshman. 1992. Parallel simulation of high-speed network multiplexors. Technical report, College of Computing, Georgia Tech.

Phillips, C. I. and L. G. Cuthbert. 1991. Concurrent discrete-event simulation tools. *IEEE Journal on Selected Areas in Communications*, 9(3):477–485, April.

Presley, M. T., P. L. Reiher, and S. Bellenot. 1990. A Time Warp implementation of shark's world. In *Proceedings of the 1990 Winter Simulation Conference*, pages 199–203, New Orleans, December.

Reed, D. A., A. D. Malony, and B. D. McCredie. 1988. Parallel discrete event simulation using shared memory. *IEEE Transactions on Software Engineering*, 14(4):541–553, April.

Reinhardt, S. K., M. D. Hill, J. R. Larus, A. R. Lebeck, J. C. Lewis, and D. A. Wood. 1993. The Wisconsin wind tunnel: Virtual prototyping of parallel computers. In *Proceedings of the 1993 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, volume 21, pages 48–60, June.

Reynolds, Jr., P. F. 1988. A spectrum of options for parallel simulation. In *1988 Winter Simulation Conference Proceedings*, pages 325–332, December.

Rich, D. O. and R. E. Michelsen. 1991. An assessment of the Modsim/TWOS parallel simulation environment. In *1991 Winter Simulation Conference Proceedings*, pages 509–518, December 1991.

Richter, R. and J. C. Walrand. 1989. Distributed simulation of discrete event systems. *Proceedings of the IEEE*, 77(1):99–113, January.

Roberts, J. W., editor. 1992. *Performance Evaluation and Design of Multiservice Networks*. Commission of the European Communities, Luxembourg.

Sokol, L. M., D. P. Briscoe, and A. P. Wieland. 1988. MTW: a strategy for scheduling discrete simulation events for concurrent execution. In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 19, pages 34–42. SCS Simulation Series, July.

Soulé, L. and A. Gupta. 1991. An evaluation of the Chandy-Misra-Bryant algorithm for digital logic simulation. *ACM Transactions on Modeling and Computer Simulation*, 1(4):308–347, October.

Sporrer, C. and H. Bauer. 1993. Corolla partitioning for distributed logic simulation of VLSI-circuits. In 7^{th} *Workshop on Parallel and Distributed Simulation*, volume 23, pages 85–92. SCS Simulation Series, May.

Steinman, J. 1992. Speedes:an approach to parallel simulation. In 6^{th} *Workshop on Parallel and Distributed Simulation*, volume 24, pages 75–84. SCS Simulation Series, January.

Su, W. K. and C. L. Seitz. 1989. Variants of the Chandy-Misra-Bryant distributed discrete-event simulation algorithm. In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 21, pages 38–43. SCS Simulation Series, March.

Tallieu, F. and F. Verboven. 1991. Using Time Warp for computer network simulations on Transputers. In *Proceedings of the 24th Annual Simulation Symposium*, volume 21, pages 112–117. IEEE Computer Society Press, April.

Thomas, G. S. and J. Zahorjan. 1991. Parallel simulation of performance petri nets: Extending the domain of parallel simulation. In *1991 Winter Simulation Conference Proceedings*, pages 564–573, December.

Turner, S. and M. Xu. 1992. Performance evaluation of the bounded Time Warp algorithm. In 6^{th} *Workshop on Parallel and Distributed Simulation*, volume 24, pages 117–128. SCS Simulation Series, January.

Wang, J. J. and M. Abrams. 1992. Approximate time-parallel simulation of queueing systems with losses. In *Proceedings of the 1992 Winter Simulation Conference*, pages 700–708.

Wieland, F. and D. R. Jefferson. 1989. Case studies in serial and parallel simulation. In *Proceedings of the 1989 International Conference on Parallel Processing*, volume 3, pages 255–258, August.

Wieland, F., L. Hawley, A. Feinberg, M. DiLorento, L. Blume, P. Reiher, B. Beckman, P. Hontalas, S. Bellenot, and D. R. Jefferson. 1989. Distributed combat simulation and Time Warp: The model and its performance. In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 21, pages 14–20. SCS Simulation Series, March.

Yu, Q., D. Towsley, and P. Heidelberger. 1989. Time-driven parallel simulation of multistage interconnection networks. In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 21, pages 191–196. SCS Simulation Series, March.