PARALLEL SIMULATION

Rassul Ayani
Department of Teleinformatics, Computer Systems Laboratory
Royal Institute of Technology (KTH)
Stockholm, Sweden

Abstract

This tutorial surveys various approaches to executing discrete event simulation programs on a parallel computer. The tutorial is focused on *asynchronous* simulation programs where different processes may advance asynchronously in simulated time. Parallelization of discrete event simulation programs requires adequate synchronization scheme. We review several synchronization schemes that have appeared in the literature in recent years. The performance result of these schemes will be surveyed and some application areas will be discussed.

1. Introduction

The analysis of large and complex systems by analytical techniques is often very difficult. The availability of low cost microcomputers has introduced simulation to many real life applications. Simulation of a system may have several objectives, including: (i) understanding behavior of a system; (ii) obtaining estimates of performance of a system; (iii) guiding the selection of design parameters; (iv) validation of a model. Simulation has been used in many areas, including manufacturing lines, communication networks, computer systems, VLSI design, design automation, air traffic and road traffic systems, among others.

Two separate classes of methodologies, called *continuous* time and *discrete* time simulation, have emerged over the years and are widely used for simulating complex systems. As the terms indicate, in a continuous simulation changes in the state of the system occur continuously in time, whereas in a discrete simulation changes in the system take place only at selected points in time. Thus, in a discrete-event simulation (DES) events happen at discrete points in time and are instantaneous. One kind of discrete simulation is the fixed time increment, or the *time-stepped* approach, the other kind is the *discrete-event* method. A typical DES algorithm is based on an ordered list of events, called

event-list, or future-event-set. The algorithm repeatedly performs the following steps:(1) removes the event with the minimum simulation time from the event-list, (2) evaluates the event and possibly, and (3) inserts new event(s), generated by step 2, in the event-list.

The traditional DES, as described above, is sequential. However, many practical simulations, e.g. in engineering applications, consume several of hours (and even days) on a sequential machine. Parallel computers are attractive tools to be used to reduce execution time of such simulation programs.

In practice, a simulation program is run with several parameter settings. For instance, to design a system various parameters must be tested to determine the most appropriate ones. One may suggest to run *replication* of a simulator on separate processors of a multiprocessor computer. The replication approach is reasonable, if the experiments are independent. However, in many practical situations parameters of an experiment is determined based on outcome of the previous experiments and thus the replication approach is not applicable. An alternative solution would be to parallelize a single run of a simulator.

In this tutorial, we discuss cases where several processors of a multiprocessor system cooperate to *execute a single simulation program* and complete it in a fraction of the time one processor would need. There are several approaches to parallel simulation some of which are briefly reviewed below (see e.g. [56] for more detail).

A. *Functional decomposition*: In this approach, the simulation support tasks (such as random number generation, event set processing and statistics collection) are performed by different processors. Generally, the time needed to execute different tasks is different and thus load balancing is a problem in this approach. Moreover, the number of support functions is limited and thus this method cannot use large multiprocessors efficiently.

B. *Time-stepped simulation*: In a time-stepped simulation, simulated time is advanced in fixed increments and ach process simulates its components at these fixed points. The time step must be short to guarantee accuracy of the simulation result. This method is inefficient if there occur few events at each point. A case where good speed can be obtained has been reported by Goli et al. [25]).

C. *Asynchronous parallel simulation*: In this paper, we focus on asynchronous parallel simulation, where each process maintains its own local clock and the local time of different processes may advance asynchronously.

The rest of this tutorial is organised as following: Some basic concepts are reviewed in Section 2 and the use of parallel simulation is argued in Section 3. We review conservative parallel simulation schemes in Section 4 and optimistic methods in Section 5. Some hybrid approaches are highlighted in Section 6. Finally, concluding remarks are given in Section 7.

2. Preliminaries

Real time is the actual time needed to run a simulator, whereas the occurrence time of events in the actual system is denoted by *simulated time*.

Event list is a list that contains all scheduled, but not yet processed events.

Timestamp of an event denotes the time the event occurs in the actual system.

State variables describe the state of the system. In the process of developing a simulation model, state variables are identified. The value of the state variables represent the essential features of a system at specific points in time. For instance, in a network, state variables represent queue length and waiting time, among others.

Causality error may occur if an event E_2 depends on another event E_1 in the actual system, but it is processed before E_1 in the simulation program. In the sequential DES described in Section 1, it is crucial to remove the event with the *minimum* simulation time from the event list to avoid causality error.

Speedup is defined as the time it takes to run a simulator on a uniprocessor divided by the time it takes to run a parallel version of the simulator on a multiprocessor. The main problems related to this metric are: (i) It depends on implementation of the sequential simulator. For instance, if the sequential simulator is slow the speedup is higher! (ii) It is hardware dependent, i.e., it depends on speed of both uniprocessor and multiprocessor computer being used. As discussed by Ayani and Berkman [7], it would be more accurate to define speedup as the time it takes to run the *most efficient sequential simulator* on a single processor of a multiprocessor divided by the time it takes to execute the parallel simulator on n processors of the *same* multiprocessor.

3. Why Parallel simulation?

Parallel discret event simulation (PDES) refers to the execution of a single DES program on a parallel computer. PDES has attracted a considerable number of researchers in recent years, because:

(i) It has the potential to reduce the simulation time of a DES program. This interest is partly due to the fact that a single run of a sequential simulator may require several hours or even days.

(ii) Many real life systems contain substantial amounts of parallelism. For instance, in a communication network, different switches receive and redirect messages simultaneously. It is more natural to simulate a parallel phenomenon in parallel.

(iii) From an academic point of view, PDES represents a problem domain that requires solution to most of the problems encountered in parallel processing, e.g., synchronization, efficient message communication, deadlock management and load balancing.

One of the main difficulties in PDES is synchronization. It is difficult because the precedence constraints that dictate which event must be executed before each other is, in general, quite complex and data dependent. This contrasts sharply with other areas where much is known about the synchronization at compile time, e.g. in matrix algebra [24]

The common approach to PDES is to view the system being modeled, usually referred to as the *physical system*, as a set of *physical processes* (PPs) that interact at various points in simulated time. The simulator is then constructed as a set of *logical processes* (LPs) that communicate with each other by sending timestamped messages. In this scenario, each logical process simulates a physicaly process. Each LP maintains its own logical clock and its own event list. The logical process view requires that the state variables are statically partitioned into a set of *disjoint states* each belonging to an LP. This view of the simulation as a set of communicating LPs is used by all of the simulation methods reviewed in this paper.

It can be shown that no *causality* errors occur if each LP processes events in *non-decreasing* timestamp order [43]. This requirement is known as *local causality constraint.* The local causality constraint is sufficient, but not necessary. This is not necessary, because two events occurring within the same LP may be independent of each other and thus can be processed in any order. Ahmed et al. [2] suggest an approach where independent events belonging to the same LP may be identified and processed in parallel.

Two main paradigms have been proposed for asynchronous parallel simulation: *conservative* and *optimistic* methods. Conservative approaches strictly avoid the possibility of any causality error ever occurring. On the other hand,

optimistic approaches make the optimistic assumption that messages arrive at different LPs in correct order. However, these approaches employ a detect and recovery mechanism to correct causality errors.

4. Conservative Approaches

Several conservative approaches to PDES have been proposed in the literature. These approaches are based on processing safe events. The main difference between these methods, as discussed in this section, lies in the way they identify safe events.

4.1 The Chandy-Misra Scheme

Chandy and Misra proposed one of the first conservative PDES algorithms [14]. In this method, as described by Misra [43], a physical system is modeled as a directed graph where arcs represent communication channels between nodes. Each node of the graph is called a logical process (LP). Each LP simulates a portion of the real system to be simulated and maintains a set of queues, one associated with each arc in the graph. Within each logical process, events are simulated strictly in the order of their simulated time. Interprocess communication is required whenever an event associated with one logical process wishes to schedule an event for another logical process. It is assumed that the communication medium preserves the order of messages, and that the timestamp of the messages sent along any particular arc are nondecreasing.

The method is conservative because a logical process is not allowed to process a message with timestamp t until it is certain that no messages will ever arrive with a timestamp less than t. To guarantee this, each node must select the message with the lowest timestamp that is now scheduled for the node or will be scheduled in future. If every input arc of a node has at least one unprocessed message, then the next message to be processed is simply the one with the lowest timestamp among all of the input arcs of the node. However, if any of the input arcs is empty, then the node will be *blocked* waiting for a message to arrive. The blocking mechanism is necessary, because if a node processes any message from one of its nonempty input queues, there is no guarantee that a message that arrives later to an empty input arc will have a timestamp equal or greater than the timestamp of the processed message.

There are two problems related to blocking a node: *memory overflow* and *deadlock*.

(i) Memory overflow: While a node is blocked because some of its input queues are empty, the other queues may grow, leading to an unpredictable storage requirement. For instance, consider the system shown in Figure 1. If node 1 sends most of the messages to node 4 via node 2, there may be many messages on arc (2,4) while node 4 is blocked waiting for a message on arc (3,4).

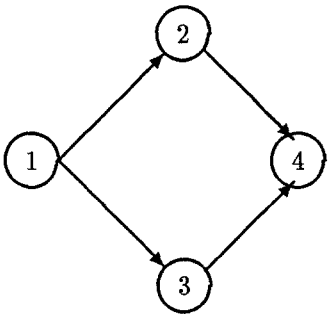

Figure 1. In this network, node 4 may cause memory overflow. For instance, node 4 may be blocked waiting for a message from node 3, while it receives lots of messages from node 2.

(ii) Deadlock: If the directed graph representing the system contains a cycle, as shown in Figure 2, then the Chandy-Misra paradigm is vulnerable to deadlock. Several methods have been proposed in the literature to resolve the deadlock problem. These methods are either based on deadlock avoidance or deadlock detection and recovery.

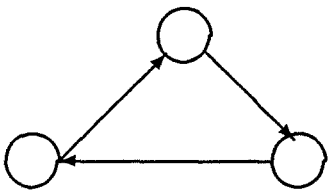

Figure 2. A circular network

4.2 Deadlock Avoidance Mechanisms

The original approach suggested by Chandy and Misra for avoiding deadlock, described in [43] is based on sending *null* messages. A null message is a dummy message used *only* for synchronoization purpose and does not correspond to any activity in the real system. A null message E_{null} with timestamp T(E_{null}) sent from LP_i to LP_j indicates that LP_i will not send any message to LP_j earlier than T(E_{null}). The receiver process may use this information to identify its next message to be processed. For instance, if LP_j is blocked waiting for a message from LP_i, this null message can be used to unblock it.

The null message scenario is straight-forward and simple to implement. However, the transmission of null messages can create substantial overhead, especially for high branching networks. One may reduce the overhead by transmitting null messages less frequently. The question is how frequently does a node need to send null messages to its successor nodes? Several mechanisms have been proposed to reduce the overhead related to null messages.

Misra [43] suggests a *time-out* scheme, where transmission of a null message is delayed for some time. This scenario decreases the total number of null messages required, because a real message with a higher timestamp may arrive or a null message with a higher timestamp may over-write the earlier null message during the time-out period. However, some processes might be delayed longer than in the original null message scheme.

Another approach is that a process sends null messages on each of its output arcs whenever it is *blocked.* It can be shown that this mechanism avoids deadlock if there is no cycle in which the collective timestamp increment of a message traversing the cycle could be zero [24].

Another approach would be to send null messages on *demand.* In this method, a process that is blocked sends a request to its predecessor asking for the earliest time the predecessor may send a message. Thus, a null message will be sent from LP_i to LP_j only when LP_j requests it. This scheme, however, may result in a cycle of requests. In this case, the message with the minimum timestamp can be processed [43].

4.3 Deadlock Detection and Recovery

Another possibility would be to let the basic Chandy-Misra schem deadlock, and provide a mechanism to detect deadlock and recover from it.

In an algorithm suggested by Misra [43], a marker circulates in a cycle of

channels. The marker is a special type of message carrying some information. The cycle is constructed in such a way that the marker traverses every channel of the network sometimes during a cycle. If an LP receives the marker it will send it to the next channel within a *finite* time. An LP is said to be *white* if it has neither received nor sent a message since the last departure of the marker from it; the LP is *black* otherwise. The marker declares *deadlock* when it finds that the last N logical processes that it has visited were all white, where N is the number of nodes in the network. As discussed in [43], the algorithm is correct if the messages communicated between the LPs are received in the time order they are sent. As an example consider the network depicted by Figure 1. The network is deadlocked if the marker visits all the three nodes and finds that all of them are white, i.e., they have neither received nor sent a message since last visit.

The marker scheme may also be used to recover from deadlock. The marker may carry the *minimum* of the next-event-times for the white LPs it visits. When the marker detects deadlock, it knows the smallest event-time and the LP at which this event occurs. To recover from deadlock, the LP with the minimum next-event-time will be restarted.

Experimental results (e.g. [22], [53], [63]) suggest that the deadlock avoidance method is superior to the deadlock detection and recovery. Unfortunately, the deadlock avoidance presumes a nonzero minimum service time.

4.4 Conservative Time Window Schemes

Several researchers have proposed window based conservative parallel simulation schemes (e.g., see [9], [38], [47]). The main idea behind all these schemes is to identify a time window for each logical process such that events within these windows are safe and can thus be processed concurrently. The basic constraint on such schemes is that events occurring within each window are processed sequentially, but events within different windows are independent and can be processed concurrently.

Consider a system consisting of n logical processes LP1, LP2,...,LPn. Assume that the following Conservative Time Window (CTW) parallel simulation scheme is used.

The CTW-algorithm shown in Figure 3 works in the following way:

a) In the first phase, a window Wi is assigned to each LPi such that the events occurring within Wi are independent from the events occurring in Wj, $i \neq j$.

Repeat

1) Identification phase
Assign a window Wi to LPi such that events in Wi can be processed concurrently with events in Wj, i $\neq j$.
Barrier
2) Process phase
Process the events in W1, W2, ...,Wn.
Barrier

Until (End-of-Simulation)

Figure 3. A Conservative Time Window (CTW) parallel simulation scheme.

The way independent windows are identified has been discussed elsewhere, e.g. see [9].

b) In phase 2 of the CTW-algorithm, the events occurring within each window are processed sequentially, but events within different windows are independent and can be processed concurrently.

c) Each phase of the algorithm may be executed by several processors in parallel. However, synchronization is required between the two consecutive phases. Thus, the next iteration of the CTW-algorithm will be started after processing all the time windows belonging to the current iteration.The algorithm produces a Time Window, which may be empty, for each LP. The width of the windows is calculated in each iteration of the algorithm. Figure 4 illustrates three iterations of the CTW-algorithm for simulating a system consisting of three subsystems.

Generally, different windows have different sizes and contain different number of events. In other words, there will be n windows W1, W2, ..., Wn with different widths to be assigned to m processors. The performance of the window based schemes depends heavily on how the windows are assigned to the processors. Several scheduling schemes have been proposed and evaluated in [8]. As discussed in [9], the number of non-empty windows produced in each iteration of the algorithm and the size of each one depends on features of the system being simulated, e.g. message population, network topology, and network size.

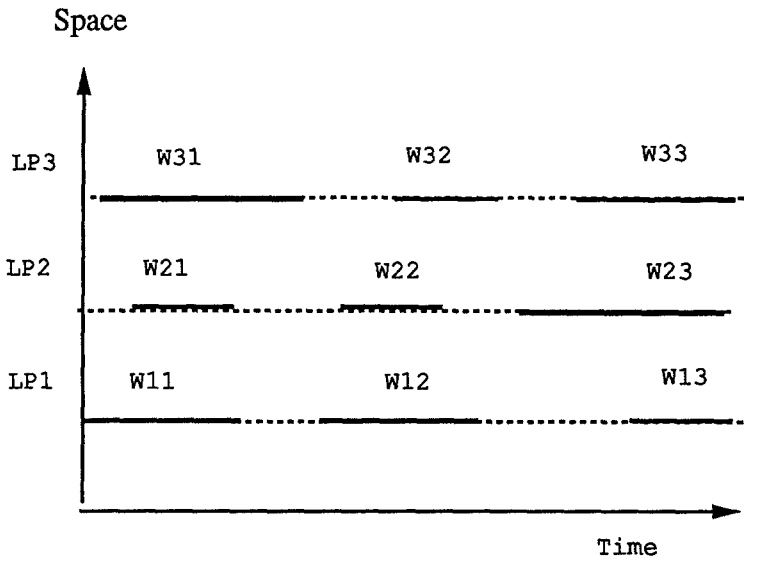

Figure 4. Results of performing 3 iterations of the CTW-algorithm on a system with three LPs, where Wij denotes the window generated for LPi in the jth iteration.

More information on the behavior of the window based parallel simulation algorithms can be found in [9].

4.5 Performance of the Conservative Schemes

Several researchers have studied the performance of the conservative schemes. The most extensive performance result has been reported by Richard Fujimoto [22], [24]. According to Fujimoto, performance of the conservative algorithms is critically related to the degree to which logical processes can look ahead into their future simulated time. Wagner and Lazowska [63] report on performance of a conservative scheme on a shared memory multiprocessor. Chandy and Sherman [16] report speedup in simulation of queueing networks. Ayani and Rajaei [9] present an intensive performance study of the conservative time window scheme on shared memory multiprocessors.

5. Optimistic Approaches

Optimistic approaches to PDES, as opposed to conservative ones, allow oc-

currence of causality error. These protocols do not determine *safe* events; instead they *detect* causality error and provide mechanisms to *recover* from such error.

The Time Warp mechanism proposed by Jefferson and Sowizral [31] is the most well known optimistic approach. The Time Warp mechanism (as described in [29]) allows an LP to execute events and proceed in its local simulated time, called local virtual time or LVT, as long as there is any messeage in its input queue. This method is optimistic because it assumes that message communications between LPs arrive at proper time, and thus LPs can be processed independently. However, it implements a roll back mechanism for the case when the assumption turns out to be wrong, i.e. if a message arrives to a node at its past. The method requires both time and space for maintaining the past history of each node, and for performing the roll back operation whenever necessary.

Under the Time Warp protocol, each message has a *send* time and a *receive* time. The send time is equal to the local clock of the sending LP when the message is sent. The receive time is the simulated time the message arrives at the receiving LP. The receive time is the same as the timestamp used in the conservative approaches. The send time concept is used to define GVT and to implement the Time Warp protocol correctly. Global virtual time (GVT) is the minimum of all LVTs and the send times of all messages that have been sent but not yet received.

If messages arrive to a process with receive times greater than the receiver's LVT, they are enqueued in the input queue of the receiver LP. However, if an LP receives an event message that "should" have been handled in its simulated past, i.e., its receive time is less than the receiver's LVT (such a message is called a *straggler*), then the receiving LP is rolled back to the simulation time before the timestamp of the straggler message. In addition to rolling back the receiving LP, however, the Time Warp mechanism must cancel all of the indirect side effects caused by any messages the receiving LP sent with timestamps greater than the time at which it is rolled back. This is done by sending *antimessages* to annihilate the corresponding *ordinary* messages.

In Time Warp, no event with timestamp smaller than GVT will ever be rolled back. Thus, all events with timestamp less than GVT can be committed and the memory space occupied by state variables up to GVT can be released. The process of committing events and reclaiming memory is referred to as *fossil collection* [23] and [24].

5.1 Lazy Cancellation

Several schemes for *undoing* side effects caused by erroneous messages have appeared in the literature. In the *aggressive cancellation* mechanism, when a process rolls back *antimessages* are sent immediately to cancel erroneous messages.

In *lazy cancellation* [23], antimessages are not sent immediately after rollback. Instead, the rolled back process resumes execution of events from its new LVT. If the reexecution of the events regenerates the *same* message, there is no need to cancel the message. Only messages that are different from the old messages are transmitted; after the process' clock passes time T, antimessages are sent only for those messages with timestamp less than T that are not regenerated. Under aggressive cancellation, a process may send unnecessary antimessages. Under lazy cancellation there are no unnecessary antimessages. However, lazy cancellation may allow erroneous computation to spread further because antimessages are sent later. 196z The lazy cancellation mechanism may improve or degrade performance of the time warp depending on features of the application. Most of the performance results reported in the literature suggest that lazy cancellation improves performance. However, one can construct cases where lazy cancellation is much slower than aggressive cancellation [23].

5.2 Performance of the Optimistic Schemes

Several researchers have report successes in using Time Warp to speedup simulation problems. Fujimoto has reported significant speedup for several queueing networks [23]. Some researchers have developed analytical models to evaluate performance of Time Warp. Analytical models for the case of two processors have been developed by Mitra and Mitrani [44], and Feldman and Kleinrock [20]. Models for multiprocesses have been developed by Akyildiz et al. [4] and Gupta et al. [26], among others.

6. Hybrid Approaches

The deadlock handling is the main cost factor in conservative methods. In optimistic approaches, the detection and recovery of causality errors require state saving and rollback . State saving may require a considerable amount of memory if system state consists of many variables that must be saved frequently. The memory requirement may be reduced if the GVT is more frequently computed and the unnecessary states are removed, i.e. the fossil collection procedure is

done more frequently. However, this requires more CPU time.

It seems reasonable to combine the advantages of these two approaches in a hybrid protocol. The issue of combining the two approaches has received considerable attention in recent years, since the limitations of each paradigm are better understood. It is believed that the future PDES paradigm will be a hybrid one!

There are three general categories of hybrid approaches:

(i) To add optimism into a conservative approach. For instance, in the speculative simulation method proposed by Horst Meh [42] whenever an LP is to be blocked, it optimistically simulates the events in its event list, but keeps the result locally until it becomes committed. In the Filtered Rollback proposed by Lubachevsky [39], the upper-bound is set to a larger value than the one determined by the the conservative bounded-lag algorithm . These hybrid schemes are still conservative and thus cannot support *dynamic* configuration of LPs.

(ii) To add conservatism to an optimistic approach. One may try to bound the advancement of LVTs in Time Warp. This technique reduces rollback frequency and the rollback distance in general. However, it tends to reduce the degree of available parallelism as well.

The main problem with this category of schemes is how to determine a boundary for limiting the optimism. For instance, in MIMDIX [40] special processes, called Genie processes, are introduced to compute upper bounds for the advancement of LVTs. In Wolf [41], whenever a rollback occurs, special messages are broadcasted to limit propagation of the erroneous messages. The bounded time warp (BTW) proposed by Turner and Xu [61] divides the simulation duration time interval into a number of equal intervals and all events within an interval is processed before the next one is started. Reiher and Jefferson [55] propose a window-based throttling scheme, where LPs are prevented from executing events in the far future. The local time warp (LTW) proposed by Rajaei [52] partitions the system into a set of clusters each containing a number of LPs. The LPs within each cluster are synchronized by Time Warp, whereas the inter-cluster synchronization is based on the conservative time window scheme described in [9].

(iii) Switching between Optismism and Conservatism. Some researchers, e.g. [5], suggest to switch between the conservative and the optimistic schemes. This approach is attractive, especially when the behavior of the application changes dynamically.

7. Conclusions

The state of the art in PDES has advanced very rapidly in the recent years and much more is known about the potentials of the parallel simulation schemes. In particular, the extensive performance studies conducted by several researchers have identified strengths and weaknesses of the parallel simulation schemes. In this paper, we attempted to provide an insight into various strategies for executing discrete event simulation programs on parallel computers and highlight future research directions in this field. The implementation of the event-list and its impact on performance, though important, was not covered in this tutorial. Interested readers are referred to [57], [58], [32] and other articles given in the reference list.

Conservative methods offer good potentials for certain classes of problems where application specific knowledges can be applied to exploit look ahead. Optimistic methods have had a significant success on a wide range of applications, however, reducing the state saving costs is still a research problem. The issue of combining the two approaches has received considerable attention in the recent years. It is believed that the future PDES paradigm will be based on hybrid approaches.

Acknowledgments

I would like to thank Hassan Rajaei and Eric Lin for reading a draft of the tutorial and suggesting improvements. Our research project on parallel simulation (PARSIM) supported by the Swedish National Board for Industrial and Technical Development, NUTEK (contract no. 90-01773), was the driving force behind this study. I would like to thank NUTEK for the support we received during the past five years.

References

[1] M. Abrams. The object library for parallel simulation (olps). In *1988 Winter Simulation Conference Proceedings*, pages 210–219, December 1988.

[2] H. Ahmed, L. Barriga, and R. Ayani. Parallel discrete event simulation using space-time events. *Submitted for publication.*

[3] I. F. Akyildiz, L. Chen, S. Das, R. M. Fujimoto, and R. F. Serfozo. Performance analysis of "time warp" with limited memory. Technical Report

TR-GIT-91-46, College of Computing, Georgia Institute of Technology, Atlanta, GA, October 1991.

[4] I. F. Akyildiz, L. Chen, S. R. Das, R. M. Fujimoto, and R. Serfozo. Performance analysis of time warp with limited memory. In *Proceedings of the 1992 ACM SIGMETRICS Conference on Measuring and Modeling Computer Systems*, volume 20, pages 213–224, May 1992.

[5] K. Arvind and C. Smart. Hierarchical parallel discrete event simulation in composite elsa. In 6^{th} *Workshop on Parallel and Distributed Simulation*, volume 24, pages 147–158. SCS Simulation Series, January 1992.

[6] R. Ayani. 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 1989.

[7] R. Ayani and B. Berkman. Parallel discrete event simulation on simd computers. *To appear in Journal of Parallel and Distributed Computing*, 18, 1993.

[8] R. Ayani and H. Rajaei. Event scheduling in window based parallel simulation schemes. In *Proceedings of the Fourth IEEE Symposium on Parallel and Distributed Computing*, Dec 1992.

[9] R. Ayani and H. Rajaei. Parallel simulation based on conservative time windows: A performance study. *To appear in Journal of Concurrency*, 1993.

[10] D. Baezner, G. Lomow, and B. Unger. Sim++: The transition to distributed simulation. In *Distributed Simulation*, volume 22, pages 211–218. SCS Simulation Series, January 1990.

[11] R. Bagrodia and W.-T. Liao. Maisie:A language and optimizing environment for distributed simulation. In *Distributed Simulation*, volume 22, pages 205–210. SCS Simulation Series, January 1990.

[12] B. Berkman and R. Ayani. Parallel simulation of multistage interconnection networks on a SIMD computer. In *Advances in Parallel and Distributed Simulation*, volume 23, pages 133–140. SCS Simulation Series, January 1991.

[13] A. Boukerche and C. Tropper. A performance analysis of distributed simulation with clustered processes. In *Advances in Parallel and Distributed Simulation*, volume 23, pages 112–124. SCS Simulation Series, January 1991.

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

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

[16] K. M. Chandy and R. Sherman. Space, time, and simulation. In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 21, pages 53–57. SCS Simulation Series, March 1989.

[17] M. Chung and Y. Chung. 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 1991.

[18] B. Cota and R. Sargent. A framework for automatic lookahead computation in conservative distributed simulations. In *Distributed Simulation*, volume 22, pages 56–59. SCS Simulation Series, January 1990.

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

[20] R. Felderman and L. Kleinrock. Two processor Time Warp analysis: Some results on a unifying approach. In *Advances in Parallel and Distributed Simulation*, volume 23, pages 3–10. SCS Simulation Series, January 1991.

[21] R. Fujimoto. Performance of Time Warp under synthethic workloads. In *Distributed Simulation*, volume 22, pages 23–28. SCS Simulation Series, January 1990.

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

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

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

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

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

[27] P. Heidelberger and D. M. Nicol. Simultaneous parallel simulations of continuous time markov chains at multiple parameter settings. In *Proceedings of the 1991 Winter Simulation Conference*, pages 602–607, 1991.

[28] P. Heidelberger and H. S. Stone. Parallel trace–driven cache simulation by time partitioning. In *Proceedings of the 1990 Winter Simulation Conference*, pages 734–737, 1990.

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

[30] D. R. Jefferson, B. Beckman, F. Wieland, L. Blume, M. DiLorento, P. Hontalas, P. Reiher, K. Sturdevant, J. Tupman, J. Wedel, and H. Younger. The Time Warp Operating System. *11th Symposium on Operating Systems Principles*, 21(5):77–93, November 1987.

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

[32] D. W. Jones. An empirical comparison of priority-queue and event-set implementations. *Communications of the ACM*, 29(4):300–311, Apr. 1986.

[33] Y.-B. Lin and E. D. Lazowska. Exploiting lookahead in parallel simulation. *IEEE Transactions on Parallel and Distributed Systems*, 1(4):457–469, October 1990.

[34] Y.-B. Lin and E. D. Lazowska. Optimality considerations of "Time Warp" parallel simulation. In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 22, pages 29–34. SCS Simulation Series, January 1990.

[35] Y.-B. Lin and E. D. Lazowska. A study of Time Warp rollback mechanisms. *ACM Transactions on Modeling and Computer Simulation*, 1(1):51–72, January 1991.

[36] W. M. Loucks and B. R. Preiss. The role of knowledge in distributed simulation. In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 22, pages 9–16. SCS Simulation Series, January 1990.

[37] B. D. Lubachevsky. Bounded lag distributed discrete event simulation. In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 19, pages 183–191. SCS Simulation Series, July 1988.

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

[39] B. D. Lubachevsky, A. Shwartz, and A. Weiss. Rollback sometimes works ... if filtered. In *1989 Winter Simulation Conference Proceedings*, pages 630–639, December 1989.

[40] V. Madisetti, D. Hardaker, and R. Fujimoto. The mimdix operating system for parallel simulation. In 6^{th} *Workshop on Parallel and Distributed Simulation*, volume 24, pages 65–74. SCS Simulation Series, January 1992.

[41] V. Madisetti, J. Walrand, and D. Messerschmitt. Wolf: A rollback algorithm for optimistic distributed simulation systems. In *1988 Winter Simulation Conference Proceedings*, pages 296–305, December 1988.

[42] H. Mehl. Speedup of conservative distributed discrete-event simulation methods by speculative computing. In *Advances in Parallel and Distributed Simulation*, volume 23, pages 163–166. SCS Simulation Series, January 1991.

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

[44] D. Mitra and I. Mitrani. Analysis and optimum performance of two message passing parallel processors synchronized by rollback. In *Performance '84*, pages 35–50, Elsevier Science Pub., (North Holland), 1984.

[45] D. Nicol, A. Greenberg, B. Lubachevsky, and S. Roy. 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 1992.

[46] D. M. Nicol. Parallel discrete-event simulation of FCFS stochastic queueing networks. *SIGPLAN Notices*, 23(9):124–137, September 1988.

[47] D. M. Nicol. Performance bounds on parallel self-initiating discrete-event simulations. *ACM Transactions on Modeling and Computer Simulation*, 1(1):24–50, January 1991.

[48] J. K. Peacock, J. W. Wong, and E. G. Manning. Distributed simulation using a network of processors. *Computer Networks*, 3(1):44–56, February 1979.

[49] B. R. Preiss. The Yaddes distributed discrete event simulation specification language and execution environments. In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 21, pages 139–144. SCS Simulation Series, March 1989.

[50] H. Rajaei and R. Ayani. Language support for parallel simulation. In 6^{th} *Workshop on Parallel and Distributed Simulation*, volume 24, pages 191–192. SCS Simulation Series, January 1992.

[51] H. Rajaei and R. Ayani. Design issues in parallel simulation languages. *To appear in IEEE Design and Test of Computers*, Sep 1993.

[52] H. Rajaei, R. Ayani, and L.-E. Thorelli. The local time warp approach to parallel simulation. In 7^{th} *Workshop on Parallel and Distributed Simulation*, January 1993.

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

[54] P. L. Reiher and D. Jefferson. Dynamic load management in the Time Warp Operating System. *Transactions of the Society for Computer Simulation*, 7(2):91–120, June 1990.

[55] P. L. Reiher, F. Wieland, and D. R. Jefferson. Limitation of optimism in the Time Warp Operating System. In *1989 Winter Simulation Conference Proceedings*, pages 765–770, December 1989.

[56] R. Righter and J. C. Walrand. Distributed simulation of discrete event systems. *Proceedings of the IEEE*, 77(1):99–113, Jan. 1989.

[57] R. Ronngren, R. Ayani, R. Fujimoto, and S. Das. Efficient implementation of event sets in time warp. In *Workshop on Parallel and Distributed Simulation (PADS)*, volume 23, pages 101–108. SCS Simulation Series, May 1993.

[58] R. Ronngren, J. Riboe, and R. Ayani. Fast implementation of the pending event set. In *International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. SCS Simulation Series, Jan 1993.

[59] L. Sokol and B. Stucky. MTW:experimental results for a constrained optimistic scheduling paradigm. In *Distributed Simulation*, volume 22, pages 169–173. SCS Simulation Series, January 1990.

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

[61] S. Turner and M. Xu. 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 1992.

[62] D. B. Wagner and E. D. Lazowska. Parallel simulation of queueing networks: Limitations and potentials. In *Proceedings of 1989 ACM SIGMETRICS and PERFORMANCE '89*, volume 17, pages 146–155, May 1989.

[63] D. B. Wagner, E. D. Lazowska, and B. N. Bershad. Techniques for efficient shared-memory parallel simulation. In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 21, pages 29–37. SCS Simulation Series, March 1989.