Proceedings of the 1992 Winter Simulation Conference
ed. J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson

STATE OF THE ART IN PARALLEL SIMULATION

Richard Fujimoto

College of Computing
Georgia Institute of Technology
Atlanta, Ga 30332-0280

David Nicol

Dept. of Computer Science
College of William & Mary
Williamsburg, VA 23187

ABSTRACT

This tutorial surveys topics that presently define the state of the art in parallel simulation. Included in the tutorial are discussions on hardware support for parallel simulation, load balancing algorithms, dynamic memory managment for optimistic synchronization, new protocols, mathematical performance analysis, and time parallelism.

1 HARDWARE SUPPORT

Hardware support for parallel discrete event simulation have been discussed in the literature for some time. Machines have been developed for simulation of logic circuits (e.g., see (Franklin *et al.*, 1984, Corporation, 1983, Pfister, 1982)), however these usually do not allow concurrent execution of events containing different timestamps.

Closer to the spirit of the parallel simulation problems discussed here is the work of Georgiadis et al. who describe a multiprocessor implementation for Simula programs (Georgiadis *et al.*, 1981). A special purpose parallel simulation engine is envisioned that utilizes a controller processor to manage the execution of the parallel simulator, and determine which processes may be executed in parallel. More recently, Concepcion describes a hierarchical, bus-based multiprocessor system for discrete event simulation applications (Concepcion, 1989). The simulator is specified hierarchically, and is then mapped directly onto a hierarchical machine architecture.

Some work has also studied hardware support for optimistic protocols. In Time Warp, processes must periodically checkpoint their state in case a rollback later occurs. State saving overheads can incur a significant overhead (Fujimoto, 1989a). One can alleviate this overhead to some extent by reducing the frequency of checkpointing, however, analytic and experimental data suggest that the optimal checkpoint interval may be frequent (e.g., every few events), which limits the utility of infrequent state saving.

Fujimoto, et al. propose a component called the *rollback chip* that provides hardware support for state saving and rollback in Time Warp (Fujimoto *et al.*, 1992). The rollback chip can be viewed as a special memory management unit. A process may issue a "mark" operation to indicate that the state of a data segment must be preserved in case a rollback later occurs. The rollback chip hardware then modifies the addresses of subsequent memory writes to preserve this data. Simulations indicate that state saving overhead can be reduced to only a few percent of the computation. A prototype implementation of the rollback chip has been developed in the commercial sector (Buzzell *et al.*, 1990). The rollback chip work has also been extended to support a timestamp addressed memory system called *space-time memory* which is the principal component of a machine architecture called the *Virtual Time Machine* that uses rollback as the principal primitive for synchronization (Fujimoto, 1989b, Ghosh and Fujimoto, 1991).

Reynolds has proposed a hardware mechanism to rapidly collect, operate on, and disseminate synchronization information throughout a parallel simulation system (Reynolds, Jr., 1991, Pancerella, 1992). The hardware is configured as a binary tree, with a processor assigned to each node. For example, simulations indicate that the time required to compute GVT is reduced by orders of magnitude over software based approaches. A prototype system is currently under construction.

2 LOAD BALANCING

The simulation contains some initial set of logical processes. New processes may be created, or existing processes deleted as the simulation progresses. Ideally, these processes should be distributed across the parallel processor so that (1) all processors remain busy doing useful work all of the time, and (2) interprocessor communications is minimized.

Static load balancing algorithms distribute a fixed set of processes over the processors in the system. *Dynamic* algorithms allow processes to migrate during the execution of the parallel simulation. Dynamic algorithms are more appropriate if (1) information to achieve proper load balancing is not available until runtime, or (2) the proper distribution of processes to processors changes dynamically throughout the program's execution, e.g., for simulations that move through phases where a load distribution appropriate for one phase is inappropriate for another.

Load balancing has been widely studied for general (i.e., not necessarily simulation) parallel and distributed computation. Many of the techniques that have been proposed (e.g., simulated annealing, numerical techniques, node exchange heuristics, pressure based load migration) can be applied to parallel simulation programs. For instance, Nandy and Loucks use an iterative, static load balancing algorithm for parallel simulation using the Chandy/Misra/Bryant synchronization protocol (null messages) (Nandy and Loucks, 1992). The algorithm begins with an initial, random, partitioning of the task graph, and then continually evaluates possible movement of nodes (logical processes) from one partition to another. This algorithm assumes much is known about the simulation in terms of computation and communication requirements of logical processes.

Early work on static and dynamic load balancing is found in (Nicol and Reynolds, 1985, Nicol, 1985). The basic idea behind the static mapping algorithm is to examine the critical paths through multiple executions of a simulation, and cluster in such a way that the critical paths are left as undisturbed as possible. The dynamic load balancing algorithm is actually dynamic invocation of the static algorithm, based on a statistical decision process that monitors the simulation's behavior and triggers a remapping when it is probable that the resulting performance gains exceed the total remapping cost.

Prior to developing their own dynamic load balancing mechanisms, the JPL TWOS group performed static load balancing for their Time Warp programs by first collecting a trace of the program's execution. Based on this trace, a task graph showing all dependencies between events is constructed, and a bin packing algorithm used to determine a suitable assignment of processes to processors. The "off-line" nature inherent in this approach led them to develop and rely upon dynamic load management algorithms instead.

Optimistic synchronization mechanisms introduce a new wrinkle to dynamic load balancing: high processor utilization does not necessarily imply good performance, because a processor may be busy executing workload that is later undone. To address this issue, Reiher and Jefferson propose a new metric called *effective processor utilization* which is defined as the fraction of the time during which a processor is executing computations that are eventually committed (Reiher and Jefferson, 1990). They propose a strategy that migrates processes from processors with high effective utilization to those with low utilization. An algorithm that is similar in spirit is proposed in (Glazer, 1992). This algorithm allocates virtual timeslices to processes, based on their observed rate of advancing the local simulation clock. Uniprocessor simulation studies reveal scenarios in which this timeslicing approach achieves significantly better performance than the Reiher and Jefferson algorithm (as much as 33%), and others where the performance of the two methods is comparable.

A second problem in Time Warp is the fact that process migration may be very expensive because processes contain a large amount of history information. Reiher and Jefferson propose splitting a process into *phases* when the process migrates to another processor. Each phase spans a contiguous segment of simulated time that does not overlap with other phases. When migration occurs, the old phase (and its corresponding history information) remain on the original processor, and the new phase begins at the new processor. Rollbacks may span phase boundaries. Reiher and Jefferson demonstrate that phase splitting and the effective utilization metric are useful to dynamically balance the load in simulations of a communication network, a system of colliding pucks, and a combat models.

3 MEMORY MANAGEMENT

While the analyses discussed above are primarily concerned with time performance, a related question is that of memory performance. A growing body of research examines storage utilization of parallel simulations.

Optimistic mechanisms maintain information concerning the history of the program's execution in order to enable recover from synchronization errors. In Time Warp, for instance, each process maintains past state vectors in its state queue, processed events in its input queue, and records of previously sent messages (anti-messages) in its output queue. A mechanism called *fossil collection* is provided to reclaim "old" history information that is no longer needed (Jefferson, 1985). Fossil collection relies on the computation of a quantity called *global virtual time* (GVT), which represents a lower bound on the timestamp of any future rollback. GVT may usually be computed as the minimum timestamp of any unprocessed or partially processed message in the system, though for certain protocols, e.g., message sendback, GVT must be computed using the timestamp of the sending process when the message was generated. Storage used by message buffers and snapshots of process state that are older than (GVT) can be reclaimed and used for other purposes. Even with fossil collection, however, the amount of storage that is required to execute Time Warp programs may be large.

How can a Time Warp program reduce its memory requirements? One approach that economizes on memory for state vectors is to reduce the frequency of state saving. The drawback of this approach is that rollbacks become more costly. To roll back to simulated time T, a process must (1) roll back to the most recent state vector older than T, and (2) recompute forward again to reach simulated time T. Message sending must be "turned off" during the recomputation phase or a domino effect could occur that rolls back the simulation beyond GVT. Infrequent state saving increases the cost of each rollback because on average, the length of each rollback is greater, and the number of events in each recomputation phase is increased. This is problematic because as illustrated in (Fujimoto, 1990), the computation is more prone to unstable execution if rollback costs are high.

Although infrequent state saving increases rollback overhead, it also decreases the time required to perform state saving, which can be substantial. This tradeoff suggests that there may be an optimal state saving frequency that balances state saving overhead and recomputation costs. This question has been studied in the context of fault tolerant computation, e.g., see (Chandy, 1975, Gelenbe, 1979). More recently, Lin and Lazowska considered this tradeoff in the context of Time Warp programs, and show that an error in overestimating the state saving frequency is more costly than an equal magnitude error in underestimating the frequency, i.e., it is better to err on the side of less-frequent-than-optimal state saving in order to maximize performance (Lin and Lazowska, 1990b). Preiss, MacIntyre, and Loucks (Preiss *et al.*, 1992) and Bellenot (Bellenot, 1992) validate this observation experimentally. Bellenot also observes that benefits in reducing state saving frequency diminish or become liabilities as the number of processors is increased.

Even with infrequent state saving, however, the question remains: what happens if the Time Warp program runs out of memory, and no additional storage can be reclaimed via fossil collection? Several approaches have been developed to address this concern. The basic idea behind these mechanisms is to roll back overly optimistic computations, and reclaim the memory they use for other purposes. Jefferson first proposed a mechanism called *message sendback* to achieve this effect (Jefferson, 1985). In message sendback, the Time Warp executive may return a message to its original sender without ever processing it, and reclaim the memory used by the message. Upon receiving the returned message, the sender will (usually) roll back to the send-timestamp of the message (i.e., the virtual time of the sender of the message when it was generated), and regenerate it. This rollback causes anti-messages to be sent (assuming aggressive cancellation), and the subsequent annihilations release additional memory resources in the system. Only messages with send timestamp greater than GVT can be returned, since otherwise, a rollback beyond GVT might result.

Jefferson's original proposal invokes message sendback when a process receives a message, but finds that there is no memory available to store it (Jefferson, 1985). The message with the largest send-timestamp is returned. Gafni proposes a protocol that utilizes message sendback as well as other mechanisms to reclaim storage used by state vectors and messages stored in the output queue when a process finds that its local memory is exhausted (Gafni, 1988).

Jefferson has proposed an alternative approach called cancelback (Jefferson, 1990). While Gafni's algorithm will only discard state in the process that ran out of memory, cancelback allows state in any process to be reclaimed. Messages containing high send-timestamps are sent back to reclaim storage allocated to messages. This tends to roll back processes that are ahead of others in the simulation.

Another approach, proposed by Lin, is the *artificial rollback* algorithm (Lin, 1992). When storage is exhausted and fossil collection fails to reclaim additional memory, processes are rolled back to recover memory. The process that is the furthest ahead is rolled back to the time of the second most advanced process. This procedure is repeated until the supply of free memory reaches a certain threshold. The principal advantage of artificial rollback over cancelback is that it is simpler to implement.

Artificial rollback and cancelback have the interesting property that they are able to execute the simulation program using no more memory than that required by the sequential execution utilizing an event list. Lin refers to protocols such as these that require no more than a constant times the amount of memory required for sequential execution as *storage optimal.* This is an attractive property because it allows the

Time Warp program to execute with whatever memory is available, provided there is enough to execute the sequential version.

It is interesting to note that while Time Warp with cancelback or artificial rollback are storage optimal, certain *conservative* simulation protocols are not. Lin et al. (Lin *et al.*, 1990) and Jefferson (Jefferson, 1990) show that the Chandy/Misra/Bryant algorithm may require $O(nk)$ space for parallel simulations executing on n processors where the sequential simulation requires only $O(n + k)$ space. Further, Lin and Preiss (?) report the existence of simulations where Chandy/Misra/Bryant have exponential space complexity, and thus utilize much more storage than the sequential simulation. On the other hand, they also indicate that this algorithm may sometimes use *less* storage than that which is required by the sequential simulator. Time Warp with cancelback or artificial rollback always requires at least this much.

Of course, a Time Warp program will run very slowly if one only provides the absolute minimum amount of memory. Recently, the question of Time Warp performance as the amount of memory is varied has been studied (Akyildiz *et al.*, 1992). An analytic model validated by experimental measurements was developed. This work indicates that for homogeneous workloads, Time Warp requires relatively little memory to achieve good performance, provided fossil collection overheads do not dominate. In particular, this work indicates that four to five buffers per processor (where a buffer holds a state vector and an event) beyond the amount required for sequential execution achieves performance that is comparable to Time Warp with unlimited memory.

4 NEW PROTOCOLS

Protocol-related research remains one of the major focii of research attention. Most new work in this area can be categorized as follows.

4.1 Enhancements to CMB algorithms

Since its first proposal in the late 1970's, researchers have proposed various optimizations to the "null-message" based synchronization developed by Chandy and Misra (Chandy and Misra, 1979), and independently by Bryant (Bryant, 1977). An optimization explored in (Cai and Turner, 1990) is to have null messages carry a list of nodes visited by the message; this permits analysis that reduces the number of null messages needed to allow forward progress. The optimization explored in (Lin *et al.*, 1990) involves restructuring simulations with no apparent lookahead, so as to remove feedback loops. Reduction of null message propagation is the object of "null message cancellation", proposed in (Preiss *et al.*, 1991).

4.2 Enhancements to Time Warp

Another body of work examines optimizations to the basic Time Warp mechanism, originally proposed in (Jefferson, 1985). Some mechanisms, such as (Madisetti *et al.*, 1988) and (Madisetti *et al.*, 1992) involve optimizations to rollback mechanism; the basic idea is to perform large-scale "preventive" rollbacks quickly, rather than let rollbacks propagate in a chain-reaction. Another line of research is to constrain Time Warp's optimism, e.g., (Turner and Xu, 1992, Ball and Hoyt, 1990, Lubachevsky *et al.*, 1989). The methods vary, but the basic idea is restrain Time Warp from executing events "too far" in the future.

4.3 Protocols Based on Windows

One emerging theme in protocol research is to study protocols that constrain all concurrent simulation activity to be within some window of global synchronization time. These protocols typically compute, distribute and are controlled by global system information. In this they reflect a philosophical shift away from the roots of parallel simulation in asynchronous distributed system theory. One promising aspect of these algorithms is their relative compatibility with SIMD architectures. The algorithms studied in (Chandy and Sherman, 1989a, Nicol, 1992, Ayani, 1989, Steinman, 1991, Gaujal *et al.*, 1992) all compute a global minimum that defines a time beyond which no processor will venture until the next window "phase", but permit processors to execute concurrently up to that point. Performance of such protocols on SIMD architectures is reported in (Berkman and Ayani, 1991) and in (Gaujal *et al.*, 1992).

4.4 Application Specific Protocols

It is frequently the case that the importance of an application justifies tailoring a protocol to its special requirements and characteristics. This approach often delivers performance advantages over "general" protocols, which may suffer extra overheads to support circumstances never encountered in the application. Protocol design for digital logic networks are considered in (Su and Seitz, 1989, DeBenedictis *et al.*, 1991); protocols for simulating general continuous-time Markov chains are developed in (Heidelberger and Nicol, 1991, Nicol and Heidelberger, 1992); and protocols for timed Petri nets are considered in (Kumar and Harous, 1990, Thomas and Zahorjan, 1991) (timed Petri nets are also studied in (Nicol and Roy, 1991), but a general purpose synchronization protocol is used).

4.5 Other Protocols

Other recent protocol research includes on-going investigation of deadlock-breaking protocols (Boukerche and Tropper, 1991, Cote and Tropper, 1992), and of parallelizing priority heap operations (Prasad and Deo, 1991).

5 ANALYTIC PERFORMANCE ANALYSIS

The last three years have witnessed an explosion of papers on the analytic performance modeling of parallel simulations. A common trait among these are assumptions made for the purposes of mathematical tractability. For example, it is commonly assumed that the time-advance associated with executing an event is an exponential random variable; it is commonly assumed that when sent, a message is routed to some processor selected uniformly at random from among all processors. Markov chains of one kind or another frequently underlie these analyses. Despite obvious limitations, this ground-breaking work in analysis is exciting because it helps to shed new understanding on the potentials—and limits—of parallel simulation.

A significant body of work is devoted to comparing different synchronization algorithms. In (Felderman and Kleinrock, 1990) it is shown that the *average* performance difference between synchronous time-stepping and an optimistic asynchronous algorithm such as Time Warp is no more than a factor of $O(\log P)$, P being the number of processors. This is actually an extreme case—if the time advance distribution is bounded from above, the performance difference is no more than a factor of 2. Conditions for the optimality of Time Warp (in the absence of overhead costs) are demonstrated in (Lin and Lazowska, 1990a). An interesting asymmetry is demonstrated in (Lipton and Mizell, 1990), with examples showing that Time Warp is capable of arbitrarily better performance than the Chandy-Misra-Bryant null-message approach and a proof that the converse is not true.

The difference between a conservative windowing algorithm and Time Warp is studied in (D.M.Nicol, 1991). This analysis includes overheads for both methods, and captures the dependence of performance on lookahead. Not surprising, the results of the comparison depend on the magnitudes of the overhead costs. Exact two-processor analyses in (Felderman and Kleinrock, 1991b) and (Felderman and Kleinrock, 1992) permit a comparison of optimistic and conservative methods in this limited case; however, this style of analysis is extended to general numbers of processors in (Felderman and Kleinrock, 1991a).

Specific protocols are sometimes analyzed. (Nicol, 1992) analyzes a conservative windowing algorithm, and demonstrates the asymptotic convergence of performance to optimality. A unique analysis (based on differential equations) of a similar algorithm is found in (Steinman, 1991). (Dickens and Reynolds, Jr., 1991) also analyze a windowing algorithm. (Eick *et al.*, 1991) propose and analyzes an asynchronous relaxation algorithm for circuit-switched networks (although the analysis carries over to other problem domains).

A detailed analysis of Time Warp is found in (Gupta *et al.*, 1991); an extension to consider the effects of limited memory is considered in (Akyildiz *et al.*, 1992). Scheduling issues in Time Warp are considered in (Lin and Lazowska, 1991a); rollback is studied in (Lin and Lazowska, 1991b) and (Lubachevsky *et al.*, 1991).

6 TIME PARALLELISM

The most obvious parallelism in physical systems is due to concurrent activity among spatially separated objects, so-called *space* parallelism. It has recently been recognized that parallelism can sometimes also be found in *time*—the behavior of a single object at different points in time can be concurrently simulated. Early recognition of this fact is found in (Chandy and Sherman, 1989b), where the authors observe that simulations are fixed-point computations, and as such can be executed as asynchronous-update computations. Practical exploitation of time parallelism was established in (Greenberg *et al.*, 1991), where it was shown how certain queueing systems can be expressed as systems of recurrence relations (in the time domain), which can be solved using standard parallel prefix methods on massively parallel machines. This line of thought was continued for certain types of timed Petri nets (Baccelli and Canales, 1992). New massively parallel algorithms for less tractable recurrence equations are developed for trace-driven cache simulations (Nicol *et al.*, 1992), and for circuit-switched communication networks (Eick *et al.*, 1991, Gaujal *et al.*, 1992).

A more direct approach to time parallelism is to partition the time domain, assigning different processors to different regions of time. A processor p

assumes some initial state for the system at the beginning point of its interval, say time t, and then simulates its interval. Now the processor whose interval terminates at t may have a different final state at t than the one assumed by p. In this case a *fix-up* operation must be performed. This method will work if the cost of a fix-up is much less than the cost of resimulating the interval. Variations on this idea are found in (Heidelberger and Stone, 1990) (for LRU cache simulation), (Ammar and Deng, 1991), and (Lin and Lazowska, 1991c).

ACKNOWLEDGEMENTS

The contribution of David Nicol was supported in part by NASA grants NAG-1-1060 and NAG-1-995, NSF grants ASC 8819373 and CCR-9201195. The contribution of Richard Fujimoto was supported in part by Innovative Science and Technology contract number DASG60-90-C-0147 provided by the Strategic Defense Initiative Office and managed through the Strategic Defense Command Advanced Technology Directorate Processing Division, and NSF grant CCR-8902362.

REFERENCES

Akyildiz *et al.*, 1992. Performance analysis of Time Warp with limited memory. *Proceedings of the 1992 ACM SIGMETRICS Conference on Measuring and Modeling Computer Systems*, 20(1), May 1992.

Ammar and Deng, 1991. Time warp simulation using time scale decomposition. pages 11–24. SCS Simulation Series, Jan. 1991.

Ayani, 1989. A parallel simulation scheme based on the distance between objects. *Proceedings of the SCS Multiconference on Distributed Simulation*, 21(2):113–118, March 1989.

Baccelli and Canales, 1992. Parallel simulation of stochastic petri nets using recurrence equations. In *Proceedings of the 1992 SIGMETRICS Conference*, pages 257–258, Newport, Rhode Island, June 1992.

Ball and Hoyt, 1990. The adaptive Time-Warp concurrency control algorithm. *Proceedings of the SCS Multiconference on Distributed Simulation*, 22(1):174–177, January 1990.

Bellenot, 1992. State skipping performance with the time warp operating system. In 6^{th} *Workshop on Parallel and Distributed Simulation*,

Berkman and Ayani, 1991. Parallel simulation of multistage interconnection networks on a simd computer. pages 133–140. SCS Simulation Series, Jan. 1991.

Boukerche and Tropper, 1991. A performance analysis of distributed simulation with clustered processes. pages 112–124. SCS Simulation Series, Jan. 1991.

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

Buzzell *et al.*, 1990. Modular VME rollback hardware for Time Warp. *Proceedings of the SCS Multiconference on Distributed Simulation*, 22(1):153–156, January 1990.

Cai and Turner, 1990. An algorithm for distributed discrete-event simulation – the "carrier null message" approach. *Proceedings of the SCS Multiconference on Distributed Simulation*, 22(1):3–8, January 1990.

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

Chandy and Sherman, 1989a. The conditional event approach to distributed simulation. *Proceedings of the SCS Multiconference on Distributed Simulation*, 21(2):93–99, March 1989.

Chandy and Sherman, 1989b. Space, time, and simulation. *Proceedings of the SCS Multiconference on Distributed Simulation*, 21(2):53–57, March 1989.

Chandy, 1975. A survey of analytic models of rollback and recovery strategies. *IEEE Computer*, 8(5):40–47, May 1975.

Concepcion, 1989. A hierarchical computer architecture for distributed simulation. *IEEE Transactions on Computers*, C-38(2):311–319, February 1989.

Corporation, 1983. Zycad Corporation. *The Zycad Logic Evaluator: Product Description.* Zycad Corp., Roseville Mn., 1983.

Cote and Tropper, 1992. On distributed and pseudosimulation. In 6^{th} *Workshop on Parallel and Distributed Simulation*, DeBenedictis *et al.*, 1991. A novel algorithm for discrete-event simulation. *Computer*, 24(6):21–33, June 1991.

Dickens and Reynolds, Jr., 1991. A performance model for parallel simulation. In *Proceedings of the 1991 Winter Simulation Conference*, pages 618–626, Phoenix, AZ, December 1991.

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

Eick *et al.*, 1991. Synchronous relaxation for parallel simulations with applications to circuit-switched networks. pages 151–162. SCS Simulation Series, Jan. 1991.

Felderman and Kleinrock, 1990. An upper bound on the improvement of asynchronous versus sychronous distributed processing. Simulation Series, Jan. 1990.

Felderman and Kleinrock, 1991a. Bounds and approximations for self-initiating distributed simulation without lookhead. *ACM Trans. on Modeling and Computer Simulation*, 1(4), October 1991.

Felderman and Kleinrock, 1991b. Two processor time warp analysis: Some results on a unifying approach. pages 3–10. SCS Simulation Series, Jan. 1991.

Felderman and Kleinrock, 1992. Two processor conservative simulation analysis. In 6^{th} *Workshop on Parallel and Distributed Simulation*,

Franklin *et al.*, 1984. Parallel machines and algorithms for discrete-event simulation. *Proceedings of the 1984 International Conference on Parallel Processing*, pages 449–458, August 1984.

Fujimoto *et al.*, 1992. Design and evaluation of the rollback chip: Special purpose hardware for Time Warp. *IEEE Transactions on Computers*, 41(1):68–82, January 1992.

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

Fujimoto, 1989b. The virtual time machine. *International Symposium on Parallel Algorithms and Architectures*, pages 199–208, June 1989.

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

Gafni, 1988. Rollback mechanisms for optimistic distributed simulation systems. *Proceedings of the SCS Multiconference on Distributed Simulation*, 19(3):61–67, July 1988.

Gaujal *et al.*, 1992. A sweep algorithm for massively parallel simulation of circuit-switched networks. Technical Report ICASE Technical Report 92-30, ICASE, July 1992.

Gelenbe, 1979. On the optimum checkpoint interval. *Journal of the ACM*, 26(4):259–270, April 1979.

Georgiadis *et al.*, 1981. Towards a parallel simula machine. *Proceedings of the 8th Annual Symposium on Computer Architecture*, 9(3):263–278, May 1981.

Ghosh and Fujimoto, 1991. Parallel discrete event simulation using space-time memory. *Proceedings of the 1991 International Conference on Parallel Processing, Vol. 3*, 3:201–208, August 1991.

Glazer, 1992. D.W. Glazer. *Load Balancing Parallel Discrete-Event Simulations*. PhD thesis, McGill University, May 1992.

Greenberg *et al.*, 1991. Algorithms for unboundedly parallel simulations. *ACM Trans. on Comp. Systems*, 9(3):201–221, 1991.

Gupta *et al.*, 1991. Performance analysis of Time Warp with multiple homogenous processors. *IEEE Transactions on Software Engineering*, 17(10):1013–1027, October 1991.

Heidelberger and Nicol, 1991. Conservative parallel simuation of continuous time markov chains using uniformization. Technical Report IBM Research Report RC-16780, IBM Research Division, April 1991.

Heidelberger and Stone, 1990. Parallel trace-driven cache simulation by time partitioning. Technical Report RC 15500, IBM Research, February 1990.

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

Jefferson, 1990. Virtual time II: Storage management in distributed simulation. *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing*, pages 75–89, August 1990.

Kumar and 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 1990.

Lin and Lazowska, 1990a. Optimality considerations of "time warp" parallel simulation. Simulation Series, Jan. 1990.

Lin and Lazowska, 1990b. Reducing the state saving overhead for Time Warp parallel simulation. Technical Report 90-02-03, Dept. of Computer Science, University of Washington, Seattle, Washington, February 1990.

Lin and Lazowska, 1991a. Processor scheduling for time warp parallel simulation. pages 11–14. SCS Simulation Series, Jan. 1991.

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

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

Lin *et al.*, 1990. Conservative parallel simulation for systems with no lookahead prediction. *Proceedings of the SCS Multiconference on Distributed Simulation*, 22(1):144–149, January 1990.

Lin, 1992. Y.-B. Lin. Memory management algorithms for optimistic parallel simulation. In 6^{th} *Workshop on Parallel and Distributed Simulation*,

Lipton and Mizell, 1990. Time Warp vs. Chandy-Misra: A worst-case comparison. Simulation Series, Jan. 1990.

Lubachevsky *et al.*, 1989. Rollback sometimes works ... if filtered. *1989 Winter Simulation Conference Proceedings*, pages 630–639, December 1989.

Lubachevsky *et al.*, 1991. An analysis of rollback-based simulation. *ACM Trans. on Modeling and Computer Simulation*, 1(2):154–192, April 1991.

Madisetti *et al.*, 1988. Wolf: A rollback algorithm for optimistic distributed simulation systems. *1988 Winter Simulation Conference Proceedings*, pages 296–305, December 1988.

Madisetti *et al.*, 1992. The mimdix operating system for parallel simulation. In 6^{th} *Workshop on Parallel and Distributed Simulation*,

Reynolds, Jr., 1991. An efficient framework for parallel simulations. pages 167–174. SCS Simulation Series, Jan. 1991.

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

Nicol and Heidelberger, 1992. Optimistic parallel simuation of continuous time markov chains using uniformization. Technical Report IBM Research Report RC-17932, IBM Research Division, April 1992.

Nicol and Reynolds, 1985. D.M. Nicol and P.F. Reynolds, Jr. A statistical approach to dynamic partitioning. Simulation Series, 1985.

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

Nicol *et al.*, 1992. Massively parallel algorithms for trace-driven cache simulation. In 6^{th} *Workshop on Parallel and Distributed Simulation*,

Nicol, 1985. D.M. Nicol. *The Automated Partitioning of Simulations for Parallel Execution*. PhD thesis, University of Virginia, August 1985.

Nicol, 1992. D.M. Nicol. The cost of conservative synchronization in parallel discrete-event simulations. *Journal of the ACM*, 1992. To appear. Available as technical report 90-20 from ICASE, Mail Stop 132C, NASA Langley Research Center, Hampton, VA 23665.

Pancerella, 1992. Improving the efficiency of a framework for parallel simulations. In 6^{th} *Workshop on Parallel and Distributed Simulation*,

Pfister, 1982. The yorktown simulation engine: Introduction. In *Proc. 19th Design Automation Conference*, pages 51–54, June 1982.

Prasad and Deo, 1991. An efficient and scalable parallel algorithm for discrete-event simulation. In *Proceedings of the 1991 Winter Simulation Conference*, pages 652–660, Phoenix, AZ, December 1991.

Preiss *et al.*, 1991. Null message cancellation in conservative distributed simulation. pages 33–38. SCS Simulation Series, Jan. 1991.

Preiss *et al.*, 1992. On the trade-off between time and space in optimistic parallel discrete-event simulation. In 6^{th} *Workshop on Parallel and Distributed Simulation*,

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

Steinman, 1991. Speedes:synchronous parallel environment for emulation and discrete event simulation. pages 95–103. SCS Simulation Series, Jan. 1991.

Su and Seitz, 1989. Variants of the Chandy-Misra-Bryant distributed discrete-event simulation algorithm. *Proceedings of the SCS Multiconference on Distributed Simulation*, 21(2):38–43, March 1989.

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

Turner and Xu, 1992. Performance evaluation of the bounded time warp algorithm. In 6^{th} *Workshop on Parallel and Distributed Simulation*,

AUTHOR BIOGRAPHIES

RICHARD FUJIMOTO is an associate professor in the College of Computing at the Georgia Institute of Technology. He received B.S. degrees in Computer Science and Computer Engineering from the University of Illinois, Urbana, in 1977 and 1978, and M.S. and Ph.D. degrees from the University of California, Berkeley, in 1980 and 1983, respectively. He has been actively involved in research in parallel discrete event simulation over the past six years. He has served as general, program, and associate-program chair for the annual Workshop on Parallel and Distributed Simulation (PADS), and currently chairs the steering committee for that meeting. He is also an area editor for ACM Transactions on Modeling and Computer Simulation (TOMACS).

DAVID M. NICOL received a B.A. in Mathematics from Carleton College, Northfield MN, in 1979. After 3 years in industry he attended the University of Virginia, where he received M.S. and Ph.D. degrees in Computer Science, in 1983 and 1985. He is presently an Associate Professor in the Department of Computer Science, at the College of William and Mary, Williamsburg, Virginia. He is an associate editor for the ACM's *Transactions on Modeling and Computer Simulation* and for the *ORSA Journal on Computing*, and has served as the 1990 Program Chairman and the 1991 General Chairman of the Workshop on Parallel and Distributed Simulation (PADS). His interests are in parallel simulation, performance analysis, and algorithms for mapping parallel workload.