# Parallel Discrete Event Simulation-Applications

Carl Tropper

*School of Computer Science*
*McGill University*
*Montreal,Canada*

E-mail: carl@cs.mcgill.ca

Over the past decade, techniques for parallel and distributed discrete-event simulation have been developed largely within the academic research community. The advent of cluster computing and desktop multiprocessors have now made it feasible to bring these results to bear on the simulation of large, complex engineering systems and on physical systems in a cost-effective manner. Our challenge now is to apply these techniques to real-world problems, i.e. to bring the area out of the laboratory. This special issue of JPDC is devoted to a collection of articles which are representative of the areas to which parallel and distributed simulation are now being applied. We feel that these articles represent a major change in the way in which discrete-event simulation will be done in the future. Taken together, they give us an indication of the benefits which will accrue from our increased ability to simulate problems which are not possible to approach on conventional uniprocessors

.

*Key Words:* parallel simulation, distributed simulation, discrete-event simulation

Since the advent of the computer, it has been common practice for engineers to simulate models of systems which they are building and for scientists to simulate models of physical systems which they are studying. This approach to designing complex systems is hardly new; shipbuilders in the European renaissance used wooden models of a ship under construction to guide them. With time, our appetites to simulate increasingly large and complex models have grown. Now we want to simulate models of the Internet and models of the formation of galaxies. Inevitably, the conventional approach of using a uniprocessor to execute these simulations has fallen victim to increased memory demands and the need for faster execution times. We are now led to the construction of virtual environments for simulation in which a scientist can interact with a virtual environment (e.g. a CAVE automatic virtual environment).

In order to accomodate the growing need for the simulation of larger models, it became necessary to make use of both distributed and parallel computer systems.

In the early 90's, the use of parallel machines for distributed simulations came into the vogue. More recently, the use of distributed computing in the form of clusters of workstations (e.g. Beowulf, Myrinet) and shared-memory multiprocessors have become widespread, due to their cost-effective nature.

Many of the systems which we wish to study can be modeled as discrete event systems. In such a model, we view the system state at discrete instants of time, in effect taking snapshots of the system at these instants. State variables can change only at these instants. In an event driven simulation, also known as discrete event simulation, the processing of an event can result in a change of state in the model. For example, in a computer network simulation an event can be the sending of a message from one computer to another. In a VLSI logic simulation, the arrival of a signal at a gate may be viewed as an event. In the event scheduling approach to discrete event simulation, each event has a timestamp associated with it. When an event is processed by a routine specific to the type of event, new events may be generated as a consequence of this processing. These new events may have larger timestamps, the result of adding a simulation time advance to the timestamp of the event. The events in the simulation are stored in a time-ordered queue, and are processed in order of the lowest timestamp first. Digital systems such as computer systems are naturally susceptible to this approach. Models of air traffic control systems and of automoblile road traffic belong to this genre, as do biological phenomena such as the spreading of a population or of a virus (epidemeological studies). Predictably, the military has an enormous range of war-gaming models which take the place of real wars for planning exercises.

Another major category of models is continuous models in which the state of the system changes continuously with time. The model is represented by family of differential equations (partial or ordinary) which depend upon time as a parameter. Physical phenomena are often described by continuous models, such as astrophysical models and models of chemical interactions. A VLSI circuit at the transistor level is described by a continuous model.

Finally, we come to the world of mixed models, in which part of the model is described as a discrete event model and part of it by a continuous model. A VLSI circuit model which comprises both the transistor and the logic gate perspectives gives us an example of a mixed model. It is becoming increasingly important to simulate these sorts of models.

There are a host of issues associated with modeling and simulation which we cannot dealve into (e.g. system models, model world-views and model executions) for reasons of both time and space. The reader interested in a comprehesive discussion of these issues would do well to consult [41]. Like any distributed program, a distributed simulation is composed of processes which communicate with one another. This communication may occur via shared memory in a shared memory multiprocessor or via message passing in a distributed envrionment. The processes in distributed simulation are each intended to simulate a portion of the system being modeled and are referred to as Logical Processes (LPs). An LP creates events, sends events to other LPs and receives events from other LPs in the course of a simulation. Associated with each LP are input queues used to store messages from other LPs, state variables which reflect the state of the physical process which is being modeled, and a local simulation clock. The purpose of the simulation clock is to capture the advance of simulation time at each LP.

The treatment of time in a distributed system is of fundamental importance to understanding and building distributed systems [29]. A distributed simulation poses a particularly vexing problem. In a uniprocessor, the events in a discrete event simulation are stored in a priority queue and simulated in the order of the smallest timestamp first. This is easily done in a uniprocessor because all of the events in the simulation are stored in memory. In a distributed simulation, however, the events are spread out among the processors involved in the simulation. A causality violation occurs if we simulate an event at an LP and an event generated by another LP with a smaller timestamp arrives afterwards in real time. This late arrival can easily cause incorrect results in the simulation. In a military simulation it matters in which order the two events "aim the gun" and "fire the gun" are executed.

The central problem of distributed simulation is the development of synchronization algorithms which are capable of maintaining causality and which do so with minimal overhead. This overhead can be in the form of increased memory demands and increased execution time. We proceed to provide an overview of some of these algorithms. In order to gain a comprehensive understanding of the field, the reader should consult the proceedings of the Workshop on Parallel and Distributed Simulation (PADS) published by the IEEE Computer Society. These proceedings are available from 1986 to the present date. A less painful way to become versed in the field is to acquire a recent book [23].

Two primary approaches to synchronization algorithms have been developed, the *conservative* and the *optimistic* classes of algorithms. A conservative algorithm is characterized by its blocking behavior. In a conservative simulation, if one of the input queues at an LP is empty, the LP blocks, awaiting a message from another LP. This behavior insures causality. (For a formal proof the reader can consult [20].) However, it does so a price. To begin with, when an LP suspends processing there may be an increase in the execution time of the the simulation. In addition, it is entirely possible for deadlocks to form. This occurs if a group of LPs is arranged in the form of a cycle such that each LP in the cycle is awaiting a message from its predecessor in the cycle. More generally, a deadlock occurs if a knot of LPs forms [22]. Hence, if a conservative synchronization algorithm is used in a distributed simulation, it becomes necessary to either find means to avoid deadlocks or to detect and break them. There are a plethora of algorithms for each of these approaches. We begin with the deadlock avoidance approach.

In the null message approach [20], each time an LP sends a message to a neighboring LP, it sends a message to its other neighbors containing the timestamp of the message. This causes the other neighbors to advance their LSTs. As a consequence, all of the LPs are aware of the earliest time that an event can arrive at an empty input queue. The LP can use this information to avoid blocking by comparing the smallest timestamp in all of its input queues to this value. If the smallest timestamp is larger then the earliest time that a message can arrive at any of the empty input queues then it can process the event associated with this timestamp safely. Otherwise, it must block. For a proof of the correctness of this approach, the reader may consult [20]. The success of this (and indeed any conservative) approach depends upon the *lookahead* information contained in the null message.

In general, lookahead corresponds to the ability of the underlying simulation model to predict the occurr ence of future events. [11] presents experimental evidence underlining the importance of lookahead by making use of a parameter he christens the lookahead ratio, which characterizes different service time distributions at the nodes of a queuing network simulation. The drawback of the null message approach is clearly the large number of messages which are sent. As can be imagined, much work has gone into reducing the number of null messages. One such approach is sending null messages only upon demand; an LP requests a null message from the LP which sends messages to an empty input queue [25]. In the carrier null message approach [27] lookahead information is attached to a null message.

Another form of deadlock avoidance is the conservative time window approach [17, 26] in which time windows are successively generated with the property that all of the events within each of the windows are safe to process.

The deadlock detection and breaking approach relies upon algorithms to detect deadlocks. In [38], a knot detection algorithm is used to detect the deadlock, followed by the determination of the minimal timestamp in the knot. The event bearing this minimal timestamp is then safe to execute.

The other major class of synchronization algorithm is known as optimistic, the prime example of which is Time Warp [14, 15]. In optimistic algorithms, LPs maintain one input queue. All of the events which arrive from other LPs are stored in the queue in increasing order of their timestamps and are processed without any concern for the arrival of events with smaller timestamps. If such a "straggler" event arrives at an LP, the LP restores its state just prior to the arrival of the straggler and continues with the simulation from that point. This process is known as rolling back. In order to restore previous states, the LP must checkpoint its state prior to the processing of an event. It is also necessary to cancel messages which were produced subsequent to the straggler, as they may well be incorrect. In order to do this, each LP maintains an output queue in which it stores copies of messages which it has already sent. There are two approaches to this cancellation. In aggressive cancellation, when a straggler arrives the LP sends the message copies queue once again. If the message and its copy meet in the LP's input queue, the two messages cancel, or annihiliate, one another. For this reason the copy is known as an anti-message. In the event that the anti-message arrives after the original message has been processed, the destination LP rolls back to the time of the anti-message, and sends out its own anti-messages. In the second approach, known as lazy cancellation, the LP does not immediately send out anti-messages upon the arrival of a straggler. Instead, it re-initiates the simulation from the checkpoint, and compares the messages which are generated with the messages in the output queue. Only if there is a difference, is an anti-message actually sent.

Clearly, the memory demands imposed by storing an LPs state prior to the processing of each event and keeping copies of messages which it has sent in the output queue are onerous. Hence techniques to reduce the amount of memory used by Time Warp have become very important to its success. One fundamental technique involves the computation of the smallest simulation time to which any LP in the simulation may be rolled back, known as the Global Virtual Time, or GVT. Each LP maintains its own version of local simulation time, known as the Local Virtual Time (LVT), defined to be the minimum timestamp of all unprocessed messages at the LP. GVT is the minimum of (1) the LVT values of all of the LPs in the simulation and (2) the minimum timestamp of all of the events which have been sent, but which have not been processed at a given point in real time. Since no LP can roll back to a time prior to the GVT, all of the memory allocated prior to the GVT may be released. Many GVT algorithms have been developed [12, 19, 1].

Another fundamental technique to reduce the amount of memory used is the periodic checkpointing of an LPs state, instead of after every event. Incremental checkpointing, in which portions of a checkpoint are stored incrementally decreases the state saving time at the expense of state restoration. It is particularly useful when the state size is large.

In a shared memory multiprocessor, the use of direct cancellation [10] is used to control the use of memory. Here pointers are used to link the children and the parents prior to the children's scheduling, thereby eliminating the need for anti-messages. Recently, a direct cancellation technique for distributed memory environments has been developed [37].

Memory management techniques can also be used to reclaim space from LPs so that a stalled simulation may be allowed to continue. These techniques include cancelback [15], artificial rollback [16], adaptive memory management [7] and a combination of flow control and dynamic load balancing [4].

A number of attempts to restrain unbridled optimism on the part of Time Warp have been made in order to reduce the possibility of rollbacks. [31] classifies these approaches as non-adaptive, adaptive based on local state and adaptive based upon global state. In the non-adaptive approach we first find the use of time windows, which constrain the events executed at an LP to lie within a time window having GVT as its lower edge. This limits both the frequency and length of rollbacks. Bounded Time Warp [2] and Moving Time Windows [3] are examples of this technique. A clear problem with this approach is determining the size of the time window. The breathing Time Bucket algorithm [33] makes use of local rollback only, i.e. it does not send anti-messages. Breathing Time Warp combines the Breathing Time Bucket algorithm with Time Warp by executing the first n events beyond GVT via Time Warp and switching to Breathing Time Buckets to execute the next m events. Adaptive Protocols based on local state attempt to control Time Warp's optimism by using information at available at LPs. Examples of this approach are [8, 9] in which LPs block probabilistically, relying upon forecasting methods (moving average, exponential smoothing, ARIMA) to determine the blocking probability. [13] makes use of local information to maximize simulation progress, providing a continuum of choices between optimistic and conservative protocols. Adaptive methods based upon global state, as their name implies, rely upon global state to control an LP's progress. The Near Perfect State Information Protocols [34] are a family of protocols which rely on the availability of "near perfect" information. One such protocol is the Elastic Time Algorithm [32] in which an LP is slowed down in proportion to the rate at which it moves ahead of its predecessor. [36] allows LPs close to the GVT to increase their event processing rate, while LPs further away from GVT are allowed smaller event processing rates.

In recent years the emphasis in the field of distributed simulation has shifted towards applying the synchronization techniques to techniques to real world problems. In the area of computer networks, an on-going effort is being undertaken to provide a parallel simulation of the Internet [30, 5]. The intention of the project is to provide a realistic test-bed for the development of new Internet protocols and to locate performance problems in aras such as routing algorithms and congestion control algorithms. To date, it has been necessary to do experiments with actual networks to achieve the same goals. A detailed parallel simulation provides us with an environment in which the experimental conditions can be controlled and in which it is possible to replicate experiments. Air-traffic simulation is making use of distributed simulation [39] as is the design of large manufacturing environments for VLSI wafers [40]. The simulation of VLSI circuitry provides another fruitful area. An optimistic simulation environment for VHDL circuitry is described in this issue in the paper by Martin et al, "Analysis and Simulation of Mixed Technology VLSI Systems". A similar project for Verilog is underway at McGill University. It goes without saying that providing the ability to simulate larger circuits and increasing the speed of a simulation gives an enormous competitive advantage to a circuit manufacturer. Military simulations are benefitting a great deal from advances in distributed simulation. A major goal in this area is the unification of simulations of different aspects of combat together e.g. tank warfare, close air support and infantry warfare. The inclusion of live exercises into the scenario then provides a formidable training environment. The SPEEDES simulation environment [33] is being used in major war-gaming exercises. The High Level Architecture(HLA) has been proposed as the common framework for DoD simulation applications [6]. The fundamental design guideline of the HLA is that the local time management mechanism used in each simulation federate (simulation component) should not be visible to other federates. Hence all forms of time management (event-driven, time-stepped, parallel discrete-event simulation and real-time driven simulation..) may be linked together. The field of Distributed Interactive Simulation is devoted to these advances. A counterpart to this military training simulation environment is a virtual laboratory , in which scientists interact with a simulation in a virtual environment. The scientist can make changes to a model of the system which he is studying, causing it to be rolled back to the point which is dictated by the changes and to resume its execution from that point. An example of this environment is the CAVE Automatic Virtual Environment.

The articles in this issue are representative of the applications of parallel simulation to such real-world problems. Two of the papers center around the parallel simulation of VLSI hardware. In "Distributed Simulation of Asynchronous Hardware", Theodoropoulos addresses the problem of simulating asynchronous VLSI designs, making use of a conservative (null-message) protocol. Asynchronous hardware departs from standard VLSI circuit design by eschewing the use of a clock to drive the hardware. In "Analysis and Simulation of Mixed-Technology VLSI Systems", Martin et al describe an optimistic simulation environment which is oriented towards the simulation of hardware designs containing both analog and digital components. Three of the papers are germane to real-time training simulations. One basic problem in these simulations is controlling the amount of message traffic in the simulation used to keep simulation entities aware of their relationship to one another. In fact, Data Distribution Management is one of the services provided by the HLA. Two of our papers are devoted to the problem of controlling this traffic. In "Dynamic Grid-Based Approach to Data Distribution Management", Boukerche and Roy describe a hybrid algorithm which they test by implementing in the RTI-kit, a run-time infrastructure. In "Space-based Communication Data Management" by Lee and Zeigler, the authors propose a space-based scheme for reducing message traffic, and test its performance in a DEVS/HLA simulation environment. DEVS is a formalism which supports simulation model development. In "Latency Hiding with Optimistic Computations", Hybinette and Fujimoto describe an optimistic technique for incorporating intensive computations which run slower than real-time in real-time training simulators. Its feasibility is demonstrated by incorporating it in a ballistic missile simulator. A biological application for PDES is described in "Simulating Spatially Explicit Problems on High Performance Architectures", by Deelman and Szymanski. In this paper, the authors describe a new rollback approach, demonstrating its effectiveness in a class of population dynamic models, the spreading of Lyme disease. In general, this class of problem belongs to the class of simulating cellular automata, a model which represents many physical phenomena including the galaxy formation, biological pattern formation and fluid flow. Finally, in "Compiler Optimized Simulation of Large-Scale Applications on High Performance Architecture", Adve et al describe compiler techniques which are useful in the simulation of parallel programs. The objective of these simulations is to predict the performance of large-scale scientific code on parallel architectures.

Parallel simulation has emerged from the laboratory and has an exciting future. The articles contained in this special issue of JPDC represent this beginning; we hope that our readers are convinced of the vitality and utility of this newly blossoming field.

## REFERENCES


1. H.Bauer, C.Sporrer. *Distributed logic simulation and an approach to asynchronous GVT calculation.* Proceedings of the Workshop on Parallel and Distributed Simulation, Newport Beach, California, pp 205-209.
2. Turner, S. and Xu, M.. *Performance Evaluation of the Bounded Time Warp Algorithm*, Proceedings of the SCS Multi-conference on Parallel and Distributed Simulation, volume 22, pages 117-126, 1992.
3. L. Sokol, B.Studky,V.Hwang. *MTW: A control mechanism for parallel discrete simulation*, Proc. of the 1989 Int. Conf. on Parallel Processing, vol III, pp 250-254.
4. M.Choe,C.Tropper. *Flow Control and Dynamic Load Balancing in Time Warp*, Journal of SCS, to appear
5. S.Bhatt, R.Fujimoto,A.Ogielski,K.Perumalla. *Parallel simulation techniques for large-scale networks*, IEEE Comm. Mag. 36(8) pp42-47
6. J.S.Dahmann. *The High Level Architecture and Beyond: Technology Challenges*, Proc. of the 13th Workshop on Parallel and Distributed Simulation(PADS99), pp.64-70
7. Das,S. and Fujimoto,R.. *Adaptive memory management and optimism control in Time Warp*, ACM Trans on Modeling and Computer Simulation, 7(2) pp 230-271
8. A. Ferscha. *Probabilistic adaptive direct optimism control in Time Warp*, Proc. of the 9th Workshop on Parallel and Distributed Simulation, Lake Placid, N.Y., pp. 120-129.
9. A. Ferscha/ *Adaptive Time Warp simulation of timed Petri nets*, IEEE Trans. Software Eng., 25(2), pp. 237-257.
10. R. Fujimoto. *Time Warp on a Shared Memory Multiprocessor*, Transactions of the Society for Computer Simulation, Vol. 6, No. 3, pp. 211-239, July 1989.
11. R.Fujimoto. *Performance measurements of distributed simulation strategies*, Trans. Soc. for Computer Simulation 6(2), pp89-132.
12. R.Fujimoto, M.Hybinette. *Computing Global Virtual Time in Shared Memory Multiprocessors*, ACM Trans. on Modeling and Computer Simulation, vol7, no4, pp. 425-446, Oct.1997

13. D.Hamnes, A.Tripathi. *Investigations in adaptive distributed simulation*, Proc. 8th Workshop on Parallel and Distributed Simulation, Edinburgh,UK, pp.20-23.

14. D.A. Jefferson. *Virtual Time*, ACM Transactions on Programming Languages and systems, Vol. 7, No. 3, pp. 404-425, July 1985.

15. D.A. Jefferson. *Virtual Time II: The Cancelback Protocol for Storage Management in Time Warp*, Proc. 9th Annual ACM Symposium on Principles of Distributed Computing, pp 75-90, ACM, 1990

16. Y-B Lin, E. Lazowska. *Reducing the State Saving Overhead for Time Warp Parallel Simulation*, T-R 90-02-03, Dept. of Computer Science and Engineering, Univ. Washington, Seattle, WA, 1990

17. B.Lubachevsky.*Bounded Lag Distributed Discrete Event Simulation*, Proc.1988 Dist Sim Conf, SCS Simulation Series, vol19,no3, Feb1988, pp183-191

18. D. E. Martin, R. Radhakrishnan, D. M. Rao, M. Chetlur, K. Subramani, and P. A. Wilsey. *Analysis and Simulation of Mixed-Technology VLSI Systems*, Journal of Parallel and Distributed Computing, Special Issue on Parallel and Distriubted Simulation

19. Mattern, F.. *Efficient Algorithms for Distributed Snapshots and Global Virtual Time Approximation*, Journal of Parallel and Distributed Computing, 18: 423-434, 1993.

20. K.Chandy, J.Misra. *Distributed Simulation:A Case Study in the Design and Verification of Distriuted Programs*, IEEE Trans. Software Eng S-5, Sept. 1979, pp. 440-452.

21. K.M.Chandy,J.Misra. *Asynchronous distributed simulation via a sequence of of parallel computations*, Comm. of the ACM 24(11), pp.198-156.

22. K.M. Chandy,J.Misra,L.M.Hass.*Distributed Deadlock Detection*, ACM Trans Comp Systems, vol1, no2, May 1983, pp 144-156.

23. R. Fujimoto. *Parallel and Distributed Simulation Systems*, Wiley Interscience, 2000.

24. H. Avril, C. Tropper. *Clustered Time Warp and Logic Simulation*, Proceedings of the 9th Workshop on Parallel and Distributed Simulation,1995, pp112-119.

25. J.Misra. *Distributed discrete event simulation*, ACM Computing Survey 18,1 ,March 1986, pp 39-65.

26. D.Nicol. *The cost of conservative synchronization in parallel discrete event simulations*, Journal of the ACM 40(2), pp. 304-333.

27. W.Cai,S.J.Tuner. *An algorithm for distributed discrete-event simulation-the carrier null message approach*, Proc.1990 SCS Multiconference on Distributed Simulation, San Diego, California, pp.3-8.

28. W.Cai, E.Letertre, S.J.Turner.*Dag Consistent Parallel Simulation: a Predictable and Robust Conservative Algorithm*, Proc. 11th Workshop on Parallel and Distributed Simulation (PADS97), pp 178-181, Lockenhaus, Austria, June 1997.

29. L. Lamport. *Time,clocks, and the ordering of events in a distributed system*, Comm. of the ACM, 21(7), pp 558-565.

30. D.Nicol, J.Crowe, A.Ogielski. *Modeling of the Global Internet*, Computer Science and Engineering, vol1, no1, Jan-Feb 1999, pp. 42-50.

31. B.J. Overeinder. *Distributed Eevent-driven Simulation:Scheduling Strategies and Resource Management*, doctoral dissertation, University of Amsterdam, 2000.

32. F. Quaglia. *A Scaled Version of the Elastic Time Algorithm*, Proc. of the 15th Workshop on parallel and distributed simulation, Lake Arrowhead, California, pp.157-165.

33. J. Steinman. *SPEEDES: A multiple-synchronization environment for parallel discrete-event simulation*, Int.Jnl. Computer Simulation 2(3), pp 251-286.

34. S.Srinivasan, P.Reynolds. *Elastic Time*, ACM Trans. on Modeling and Computer Simulation 8(2), pp.103-139.

35. J.Steinman. *Breathing Time Warp*, Proc. 7th Workshop on Parallel and Distributed Simulation, San Diego, California, pp, 109-118.

36. . S.Tay, Y.Teo,S.Kong. *Speculative parallel simulation with an adaptive throttle scheme*, Proc. 11th Workshop on parallel and distributed simulation, Lockenhaus,Austria, a997, pp. 117-126.

37. J-L. Zhao, C.Tropper. *The Dependence List in Time Warp*, Proc. 15th Workshop on Parallel and Distributed Simulation (PADS01), Lake Arrowhead, California, May 2001, pp.35-46.

38. A.Boukerche, C.Tropper. *Parallel Simulation on the Hypercube Multiprocessor*, Distributed Computing, Springer-Verlag, vol.8, no.4, pp 181-191.

39. F.Wieland. *Practical Parallel Simulation applied to Aviation Modeling*, Proc. 15th Workshop on Parallel and Distributed Simulation(PADS2001), Lake Arrowhead, California, pp. 109-117

40. Ye Zhang,W.Cai,S.J.Turner. *A Parallel Object-oriented Manufacturing Simulation Language*, Proc. 15th Workshop on Parallel and Distributed Simulation(PADS01), Lake Arrowhead, California, pp.101-108.

41. B.Zeigler,H.Prahoefer,T.Kim. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*, Academic Press.