The Cost of Terminating Optimistic Parallel Discrete-Event Simulations

Vasant Sanjeevan and Marc Abrams

TR 92-25

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

May 15, 1992

The Cost of Terminating Optimistic Parallel Discrete-Event Simulations

May 15, 1992
TR 92-25

Vasant Sanjeevan
Marc Abrams

Department of Computer Science
Virginia Tech
Blacksburg, VA 24061-0106

(703) 231-8457
{vasant, abrams}@cs.vt.edu

Abstract

Simulation models use many different rules to decide *when* to terminate. Parallel simulation protocols generally use a single, simple rule: each logical process terminates when it reaches a predefined time. Parallel simulation protocols can be broadly classified into two categories: conservative and optimistic. In a previous paper, we proposed seven algorithms for mechanically adding an arbitrary termination condition to a conservative-synchronous non-terminating parallel simulation. Informal arguments about the performance of each algorithm were made, and the arguments were confirmed through measurement for four of these algorithms. The benchmark used was the simulation of a torus network using the Bounded Lag protocol on a shared memory multiprocessor. In this paper, we report on the performance of the simulation of the same torus network benchmark with the same four termination conditions using an optimistic protocol on a message-passing multiprocessor. We also report on the performance of a colliding pucks simulation with three additional termination conditions.

1. Termination Algorithms for Parallel Discrete-Event Simulations

We consider the *parallel simulation termination problem*: given a non-terminating parallel simulation and a termination condition, denoted C, find some value of simulation time, denoted t, such that C evaluated using simulation attribute values at time t has value *true,* if such a time exists. In this paper, we will always assume such a time exists.

Figure 1 illustrates a portion of a space-time rectangle (Chandy and Sherman 1989). Let T denote a set which contains all simulation times at which a simulation attribute required to evaluate termination condition C changes value. Let n denote the number of simulation attributes required to evaluate C. There exists a mapping from each element t of set T to an n-tuple containing the values of the n simulation attributes, a_0 to a_{n-1}, required to evaluate C at time t. Set T contains an infinite number of elements, because the simulation is non-terminating. In the figure, attribute a_1 changes value at time $t_0 \in T$, both a_0 and a_2 change value at time $t_1 \in T$, and so on. The parallel simulation termination problem is equivalent to searching the space-time rectangle for *any* time $t \in T$ at which condition C, evaluated using the attribute values at time t, yields *true*.

Two termination algorithms are proposed in a previous paper (Abrams and Richardson 1991):

Exhaustive Termination Algorithm: Evaluate $C(t)$ at each simulation time represented by set T in ascending time order until $C(t)$=*true*.

Interval Termination Algorithm: Choose any subset of T such that the subset contains a time for which $C(t)$=*true*. Evaluate $C(t)$ at each simulation time represented by the chosen subset in ascending time order until $C(t)$=*true*.

One can construct cases to show that exhaustive termination sometimes requires less wall clock time than interval termination to identify a simulation time at which $C(t)$ is *true*, however the average case performance is that interval termination requires less wall clock time. The challenge in using interval termination is choosing a subset of T that contains a time t for which $C(t)$=*true* before execution of a simulation begins. However such a subset can always be chosen before execution for a stable condition. Consider Figures 2 and 3, which characterize a stable and a non-stable termination condition, respectively. From Figure 3, for a stable condition, interval termination works for *any* subset of T that contains a time t whose value is greater than the first time for which C evaluates to *true* for a stable condition. Examples of subsets of T that work for stable termination conditions are to evaluate the termination condition every tenth time that an attribute changes value, and to evaluate the termination condition every five seconds of wall clock time.

We first categorize termination algorithms based on whether the termination detection is done using the Exhaustive Termination algorithm (category E) or the Interval Termination algorithm (I). We also categorize termination algorithms based on whether the evaluation of termination condition C is done sequentially by one processor (S) or is itself parallelized and done by more than one processor (P). Termination algorithms are categorized by two letter mnemonics, such as ES.

The first benchmark we use to measure the performance of our termination algorithms is a simulation of an N X N torus G/G/1 queuing network for any non-preemptive service discipline (Figure 3, Lubachevsky 1989). Figure 4 illustrates a 3 x 3 network. Each server is modeled by a *Logical Process (LP)*. Jobs which leave a server are randomly routed with equal probability onto one of the four outgoing links. Initially, servers are idle and have an equal number of jobs in their queues. We chose 200 as the initial number of jobs in each server's input queue, after verifying by experimentation that

this is a large enough number to preclude the possibility of a server running out of jobs to process in its input queue. We use an exponentially distributed service time with a mean of 200 simulation time units and a constant of 100 time units added to it. We verified by experimentation that the simulation running times were not sensitive to these particular values.

The second benchmark we use is the Colliding Pucks simulation (Jefferson and Hontalas et. al. 1989). Colliding Pucks is a distributed discrete-event simulation of two-dimensional *pucks* moving on a flat surface, colliding with each other and with stationary *cushions* on the border of the simulation space. The number of pucks in the simulation is denoted by K. The flat surface is divided into equal sized *sectors* which form a grid and regulate the movement and interactions of pucks within their boundaries. Sectors are numbered row wise in ascending order starting from 0. Each sector, puck and cushion comprising the simulation is modeled by an *LP*. The implementation of Colliding Pucks employs simple kinematic and dynamic principles such as conservation of momentum and energy. Since there is total conservation of energy and no friction, the pucks remain in constant motion.

The term *simulation size* is defined as the value of N for the torus network benchmark and as the value of K for the Colliding Pucks benchmark. The number of processors used in a simulation run is denoted by P.

2. Termination Algorithms for Optimistic Protocols

In an optimistic protocol (Jefferson 1985), the set of attributes comprising the parallel simulation is partitioned among all the *LP*'s. The attributes assigned to a particular *LP* are *local* attributes of that *LP*. The *LP*'s execute in parallel, performing their computations and interacting with one another. Interaction between *LP*'s is effected through the sending and receiving of *event messages*. A message can cause an *LP* to change its local variables and send messages to other *LP*'s or itself, depending on the type

of message it receives and the data that the message contains. Every event occurs at a single instant of simulation time and is executed atomically.

All events in the simulation are ordered by *virtual time*. The *Local Virtual Time (LVT)* of an *LP* is defined as being equal to the timestamp of the set of messages that the *LP* is currently processing or waiting to process. Each message sent from an *LP* must have a *receive* timestamp, which defines the virtual time at which the destination *LP* will process the message. Messages cannot be sent backward in virtual time; thus the timestamp of a message must be greater than the *LVT* of the sending *LP*. Each *LP* maintains an input queue of all the messages scheduled for it in timestamp order. When an *LP* executing on a processor finishes processing an event, the simulation system gives control of the processor to the *LP* with the lowest *LVT*, which then processes the next message in its input queue. Since each processor schedules events without consulting any other processor, some events may be performed out of order. The optimistic simulation protocol will eventually detect such out-of-order events and *roll them back*, as described shortly. All effects of an out-of-order event, including any messages it may have caused to be sent, will be completely undone.

Rollback is effected by maintaining a history of *states* for each *LP* as the simulation proceeds. State information for an *LP* at a given virtual time consists of all local simulation attribute values at that virtual time instant. As the simulation proceeds, each *LP* periodically saves copies of its state in memory. When an *LP* needs to be rolled back to a particular virtual time, the simulation system retrieves state information for that virtual time instant from the history of states that each *LP* maintains.

Because some events may need to be rolled back, certain program actions must be delayed until we are certain they will not be rolled back. Such actions must wait for *commitment*. The simulation protocol calculates a quantity called *Global Virtual Time*

(GVT) to determine when actions can be safely committed. *GVT* is the virtual time of the earliest event currently in the simulation that might still be rolled back. The protocol calculates *GVT* periodically. When *GVT* exceeds the virtual time of an irreversible action, that action is committed. *GVT* is also used to determine which messages and states can be garbage collected, freeing memory for reuse. An *LP* with no messages waiting to be processed is considered to be at *LVT* positive infinity. When *GVT* reaches positive infinity, the simulation terminates execution.

One can design efficient termination detection algorithms which require modifications to the mechanism of the optimistic simulation protocol itself. However, the implementation of such algorithms requires non-trivial changes to the implementation of the protocol. We propose two termination detection algorithms that can be used with an existing optimistic simulation protocol. Consequently, the performance we achieve is a worst case bound on the performance of an optimistic simulation system that incorporates a termination algorithm.

We augment the parallel simulation with an *LP* designated the *Termination Detector* (*TD*). The *TD* runs in parallel with the other *LP*'s comprising the simulation. One or more of the *LP*'s periodically sends an UPDATE_EVENT message with a timestamp equal to its *LVT* to the *TD*. The determination of which *LP*'s need to send UPDATE_EVENT messages to the *TD* depends on the nature of the termination condition. The frequency with which UPDATE_EVENT messages are sent to the *TD*, as well as the number of *LP*'s sending these messages is determined by the simulation size and the nature of the termination condition. The UPDATE_EVENT message contains the local attributes of the sending *LP* which the *TD* requires to evaluate the global termination condition *C*. The *TD* maintains a copy of all simulation attributes required to evaluate *C*. When the *TD* receives an UPDATE_EVENT message, it updates its copy of all simulation attributes reported by the UPDATE_EVENT message and then evaluates *C*. Since the UPDATE_EVENT

message could conceivably be rolled back in the future, the evaluation of *C* itself is being done optimistically. If *C* evaluates to *false*, the *TD* does not perform any action. If, however, *C* evaluates to *true*, the *TD* will send a TERM_EVENT message to all the *LP*'s and to itself with the timestamp of the UPDATE_EVENT message it just received.

If the timestamp of the TERM_EVENT message sent by the *TD* is less than the *LVT* of the receiving *LP*, the receiving *LP* will roll back to a virtual time equal to the timestamp of the TERM_EVENT message, flush the remaining messages in its input queue and advance its *LVT* to positive infinity. If the *LP* receiving the TERM_EVENT message does not need to be rolled back, it will process the TERM_EVENT message and advance its *LVT* to positive infinity. The simulation ends when all processes, including the *TD*, have no messages left to process and *GVT* reaches positive infinity.

It is conceivable that the *TD* itself might need to be rolled back after it has scheduled TERM_EVENT messages for some or all *LP*'s. This could happen if the *TD* receives an UPDATE_EVENT message with a timestamp *less* than the timestamp of the message whose processing caused the evaluation of *C* to yield *true*. In this event, rollback is performed just as transparently as any other message rollback. Out-of-order TERM_EVENT messages sent by the *TD* causes receiving *LP*'s to advance their *LVT*'s to positive infinity. This is undone by rolling back all *LP*'s which processed the TERM_EVENT message to a time smaller than the timestamp of the TERM_EVENT and the simulation then proceeds as normal. Consequently, we could never have a simulation terminating as a result of optimistically scheduled TERM_EVENT messages which are rolled back in the future.

2.1 Algorithm IS

Algorithm IS is an interval termination algorithm and detects stable termination conditions. It employs a subset of *LP*'s to send an UPDATE_EVENT message to the *TD*

periodically with a user specified frequency. Whenever an *LP* sends an UPDATE_EVENT message, it includes the value of any attribute local to the *LP* required to evaluate *C*. The frequency with which UPDATE_EVENT messages are sent depends on the simulation size. Too high a frequency will overwhelm the *TD* with UPDATE_EVENT messages, and message processing at the *TD* becomes a bottleneck for the simulation. Too low a frequency is likely to cause the *TD* to detect termination at a virtual time significantly greater than the first virtual time instant that evaluation of *C* would have yielded *true*. This presents a problem as we need the values of some simulation attributes at the simulation time at which termination occurred in order to calculate simulation output measures. With this tradeoff in mind, we choose an UPDATE_EVENT message frequency such that the arrival rate of messages at the *TD* is independent of the simulation size and approximately equal to the arrival rate of messages scheduled for the *LP*'s. This is done by reducing the frequency with which UPDATE_EVENT messages are sent by individual *LP*'s as the simulation size increases and the number of *LP*'s sending UPDATE_EVENT messages increases.

2.2 Algorithm ES

Algorithm ES is exhaustive and it can detect non-stable as well as stable termination conditions. Each time the *LVT* of an *LP* advances, and any attributes required to evaluate *C* were modified since the *LVT* last advanced, the *LP* sends an UPDATE_EVENT message to the *TD* containing the values of all its modified local attributes. This is necessary to be able to detect non-stable termination conditions. This algorithm is more likely to create a bottleneck at the *TD* than algorithm IS due to the large increase in the arrival rate of messages at the *TD* as the simulation size increases.

3. Optimistic Termination Algorithms Performance

We use seven termination conditions for our experiments. Termination conditions T0, T1, T2 and T3 are used with the torus network benchmark. P0, P1, and P2 are used with the Colliding Pucks benchmark. The seven termination conditions are described below:

T0: Algorithm IS: "stop when the total number of jobs serviced by all *LP*'s *exceeds* 100,000"

T1: Algorithm ES: "stop when the total number of jobs serviced by all *LP*'s *equals* 100,000"

T2: Algorithm IS: "stop when the local virtual time of each *LP exceeds* 100,000 simulation time units"

T3: Algorithm ES: "stop when the local virtual time of any *LP first exceeds* 100,000 simulation time units"

P0: Algorithm IS: "stop when the total number of collisions in all even numbered sectors is *greater than* 500.

P1: Algorithm ES: "stop when the total number of collisions in all even numbered sectors is *equal to* 500.

P2: Algorithm ES: "stop when there are *greater than or equal to* 0.75**K* pucks in all even numbered sectors *or* the simulation time of all *LP*'s exceeds 4000 simulation time units.

P2 is a disjunct of two termination conditions. If 75% of the pucks are never in even numbered sectors, the simulation will still terminate when the simulation time exceeds 4000 simulation time units. We choose the number 0.75**K* after verifying by experimentation that this condition eventually becomes true for both the smallest and largest values of *K* that we use.

Performance is evaluated by measuring the *speedup* of the simulation as processors are added. Speedup is defined as the ratio of the execution time of a simulation executed by one processor under a sequential simulator to the execution time of the same simulation running on multiple processors under a parallel simulator. The only effect of adding processors is on the speedup of the simulation.

3.1 Predicted Performance

Algorithm IS with T0: Algorithm IS is an interval termination algorithm, and we only need to send UPDATE_EVENT messages to the *TD periodically*. Each *LP* sends one UPDATE_EVENT message to the *TD* each time it processes $N^2/10$ messages. The value $N^2/10$ is selected to ensure that the arrival rate of messages at the *TD* is independent of N and that termination is detected soon after evaluation of C would first yield *true*.

As N is increased, the simulation time at which termination is detected decreases because the number of servers processing jobs increases. Therefore, we expect a flat curve for the simulation running time as a function of N for a fixed P. Also, for a fixed N, we expect a decrease in the running time as P is increased.

Algorithm ES with T1: Algorithm ES is exhaustive, and each *LP* must send one UPDATE_EVENT message to the *TD* after *every* job it processes so that the *TD* can detect the *earliest* simulation time at which the 100,000th job in the simulation is processed. Therefore the arrival rate of messages at the *TD* grows linearly with the number of servers, N^2. This creates a bottleneck at the *TD* and should increase the wall clock time required for the simulation to terminate as compared to the time taken by an interval termination algorithm. As was the case with termination condition T0, the simulation time at which T1 holds decreases as N increases. Therefore, as N increases, each server sends the *TD* fewer messages, and this decreases the wall clock time at which the simulation terminates. Of the previous two effects, the tendency of the *TD* to be a bottleneck is dominant, and we expect a linear curve with a small positive slope for the running time of the simulation as N is

increased for a fixed P. As before, for a fixed N, we also expect a reduction in the running time as P is increased.

Algorithm IS with T2: Algorithm IS is an interval termination algorithm, and the simulation time at which T2 holds is independent of N. We only need to have *one LP* send UPDATE_EVENT messages to the *TD every* time its *LVT* changes as *LP*'s synchronize periodically for every *GVT* computation. This ensures that the arrival rate of messages at the *TD* is independent of N and thus message processing at the *TD* never becomes a bottleneck. It is however conceivable that the single *LP* we choose to send UPDATE_EVENT messages to the *TD* may occasionally pick a very large service time from the exponential distribution and schedule an event far in the future. This may degrade performance because the *LP* could repeatedly process the event far in the future, send an UPDATE_EVENT message to the *TD*, and then get rolled back every time an event with a timestamp in its logical past arrives. As N is increased for a fixed P, we expect the running time to increase linearly and thus get a curve with a small positive slope. Again, for a fixed N, we expect a reduction in the running time as P is increased.

Algorithm ES with T3: Algorithm ES is exhaustive, and again the simulation time at which T3 holds is independent of N. To be able to detect T3, *every LP* sends an UPDATE_EVENT message to the *TD every* time its *LVT* changes because we need to detect the *first* time that the *LVT* of *any LP* exceeds 100,000 simulation time units. As was the case in Algorithm ES with T1, message processing at the *TD* is a bottleneck. As N is increased for a fixed P, we expect the running time to increase either linearly with a large slope or quadratically. As before, for a fixed N, we expect a reduction in the running time as P is increased.

Algorithm IS with P0: Algorithm IS is an interval termination algorithm, and the *LP*'s modeling the pucks send UPDATE_EVENT messages to the *TD periodically*. Each *LP* modeling a puck sends an UPDATE_EVENT message to the *TD* after every 10 collisions. Since both pucks involved in a collision report a collision to the *TD*, the *TD* has to divide

its collision count by a factor of 2 to get the correct collision count. As K increases, the simulation time at which termination is detected decreases because the number of collisions per unit simulation time is proportional to K. Therefore, we expect to have a flat curve for the simulation running time as a function of K for a fixed P. Also, for a fixed K, we expect a decrease in the running time as P is increased.

Algorithm ES with P1: Algorithm ES is exhaustive, and the *LP*'s modeling each puck sends an UPDATE_EVENT messages to the *TD* after *each* collision. As in algorithm IS with P0, the simulation time at which termination is detected decreases as K increases. However, the arrival rate of messages at the *TD* might be high enough for it to become a bottleneck. Therefore, we expect to have either a flat curve or a curve with a small positive slope for the simulation running time as a function of K for a fixed P. Again, for a fixed K, we expect a decrease in the running time as P is increased.

Algorithm ES with P2: Algorithm ES is exhaustive, and the *LP*'s modeling even numbered sectors send UPDATE_EVENT messages to the *TD every* time a puck enters or leaves an even numbered sector. The *TD* accordingly increments or decrements its count of pucks in even numbered sectors. The simulation time at which termination is detected is independent of K. Therefore, we expect to have a curve with a positive slope for the simulation running time as a function of K for a fixed P. For a fixed K, we would expect a decrease in the running time as P is increased.

3.2 Measured Performance

Our simulations were implemented on a BBN Butterfly with 84 processors using JPL's Time Warp Operating System (TWOS) Version 2.7.1. We use N = 3, 4, 6, 8, 10, 16 and 20 for the four combinations IS+T0, ES+T1, IS+T2 and ES+T3 and K = 2, 4, 8, 16, 32, 64 and 128 for IS+P0, ES+P1 and ES+P2. The number of processors P used for the parallel simulation is varied from 1 to 64 through powers of 2. We also ran the simulation on JPL's TWSIM sequential simulator.

The simulations were statically load balanced for all runs using the TWOS load balancer. The load balancer calculates the assignment of *LP*'s to physical processors that minimizes overall simulation running time. For the torus network benchmark, load balancing is done by running the simulation on a sequential simulator for each different value of *N*. This generates output files containing statistics of the relative work done by each *LP* comprising the simulation for each different value of *N*. The TWOS load balancer then uses these files to generate load balanced configurations for all combinations of *N* and *P*. Load balancing for the Colliding Pucks benchmark was done in a similar manner.

Algorithm IS with T0 (Figure 5): The curves have a small positive slope, which we conjecture is due to an increase in message passing overhead as *N* increases. The parallel simulation running time decreases as *P* is increased from 1 to 32, but it increases with 64 processors. We conjecture that with 64 processors, the *TD* is close to *saturation*, when it receives messages at a rate which is almost equal to its maximum service rate. The parallel simulation requires more time than the sequential simulation when fewer than 4 processors are used. The running times are, however, sensitive to the rate with which UPDATE_EVENT messages are sent by the *LP*'s, which in this case is $N^2/10$.

Algorithm ES with T1 (Figure 6): The curves have a positive slope, as predicted. The running time decreases as *P* is increased from 1 to 8, but it becomes disproportionately large and erratic when *P* exceeds 8. As in the IS+T0 case, we conjecture that the *TD* is close to saturation at this point.

Parallel simulation of ES+T1 requires more time than sequential simulation. Therefore the cost of switching from an interval termination algorithm (IS+T0) to an exhaustive termination algorithm (ES+T1) for the torus network benchmark is to preclude any speedup.

Algorithm IS with T2 (Figure 7): The curves have a positive slope, as predicted. The running time decreases as *P* is increased from 1 to 64. The parallel simulation requires more time than the sequential simulation when fewer than 4 processors were used.

Algorithm ES with T3 (Figure 8): The curves have a large slope, as predicted. The running time decreases as P is increased from 1 to 16, but increases with 32 and 64 processors. This could again be due to the *TD* nearing saturation. Once again the sequential simulation does better than any of the parallel simulations.

Algorithm IS with P0 (Figure 9): The curves have a hump at K=16 but are otherwise essentially flat. The parallel simulation running time decreases as P is increased from 4 to 64. The parallel simulation requires less time than the sequential simulation when more than 4 processors were used. The simulation ran out of memory for P=1 and P=2. The running times are not sensitive to the rate with which UPDATE_EVENT messages are sent, which in this case is after every 10 events.

Algorithm ES with P1 (Figure 9): All the running times for this algorithm are essentially the same as those for Algorithm IS+P0. We conjecture that due to the complex nature of this simulation, the overhead due to a high message arrival rate at the *TD* is negligible compared to the rate with which messages are generated and processed by the simulation itself.

Algorithm ES with P2 (Figure 10): The curves have a positive slope, as predicted. The parallel simulation running time decreases as P is increased from 4 to 32. The parallel simulation requires less time than the sequential simulation when more than 4 processors were used. The simulation ran out of memory for P=1 and P=2.

4 Conclusions

This paper proposes four termination detection algorithms for optimistic parallel discrete-event simulations. All four algorithms employ a user process, the *TD*, to perform termination detection. We measure the performance of two of these algorithms with seven termination conditions using two benchmarks. We note that using algorithm ES precludes speedup for the torus network benchmark. We conjecture that this is due to the

disproportionately large message arrival rate at the *TD*, making message processing there a bottleneck for the entire simulation. One approach to solve this problem might be to employ a multiplicity of *TD*'s in a manner that alleviates the bottleneck caused by using only one *TD*, possibly by using combining trees. Another approach might be to make more intelligent guesses about the time intervals during which the messages are sent to the *TD* and reduce the number of messages sent to it while still being able to detect all classes of termination conditions (Abrams 1992).

We also observe that after a point, adding extra processors may actually degrade performance and conjecture that this might be due to the saturation of the *TD*.

Some issues that still need to be addressed are to see how our results would change if the IS and ES algorithms were embedded in TWOS itself. How biased are the results obtained by implementing the termination detector as a user process? Algorithm IS + T2 is already implemented in TWOS as a user facility embedded in the simulation system. However it can only detect termination at virtual time instants when *GVT* is calculated and would detect T2 the first time *GVT* is calculated after T2 becomes *true*.

Acknowledgments

We would like to thank the personnel of JPL for the use of their resources. We made extensive use of JPL's Time Warp Operating System running on a BBN Butterfly to run our simulations. We also used the Colliding Pucks simulation which is part of the Time Warp software distribution to measure the performance of our termination algorithms.

References

Abrams, M. 1992. *Terminating Parallel Simulations*. Technical Report 92-01, Computer Science Department, Virginia Tech, Blacksburg, Virginia.

Abrams, M. and Richardson, D. S. 1991. Implementing a Global Termination Condition and Collecting Output Measures in Parallel Simulation. In *Proceedings of the 1991 Workshop on Parallel and Distributed Simulation* , 86-91. Anaheim, California.

Bershad, B. N., Lazowska, E. D. and Levy, H. M. 1987. "PRESTO: A System for Object-Oriented Parallel Programming". Technical Report, Department of Computer Science, University of Washington, Seattle.

Chandy, K. M. and Misra, J. 1988. *Parallel Program Design: A Foundation,* Addison-Wesley, Reading, Mass.

Chandy, K. M. and Sherman, R. 1989. Space-Time and Simulation. In *Proceedings of Distributed Simulation 1989, 53-59.* Tampa, Florida.

Jefferson, D. R. 1985. Virtual Time. *ACM Transactions on Programming Languages and Systems* 7: 404-425.

Jefferson, D., Hontalas, P. et. al. 1989. Performance of the Colliding Pucks simulation on the time warp operating system (Part 1). In *Proceedings of Distributed Simulation 1989, 3-7.* Tampa, Florida.

Lakshman, T.V. and Wei, V.K. On efficiently Computing Functions of Distributed Information.

Lubachevsky, B. 1989. Efficient distributed event-driven simulations of multiple loop networks. *Communications of the ACM* **32**: 111-123.

Mattern, F. 1987. Algorithms for Distributed Termination Detection. *Distributed Computing* **2**: 161-175.

Nicol, D. M. 1990. The Cost of Conservative Synchronization in Parallel Discrete Event Simulations. ICASE Report No. 90-20, NASA Langley Research Center, Hampton, Virginia.

Richardson, D. S. 1991. Terminating Parallel Discrete Event Simulations. Master's Thesis, Technical Report 91-9, Computer Science Department, Virginia Tech, Blacksburg, Virginia.

Sanjeevan, V. and Abrams, M. 1991. The Cost of Terminating Synchronous Parallel Discrete-Event Simulations. In *Proceedings of the 1991 Winter Simulation Conference,* 642-651, Phoenix, AZ .

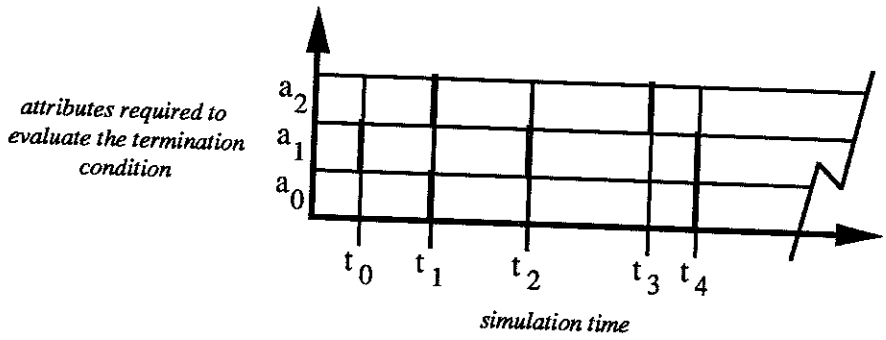

Figure 1: Illustration of space time rectangle for a simulation with three attributes. The heavy vertical lines denote assignment of new values to attributes.

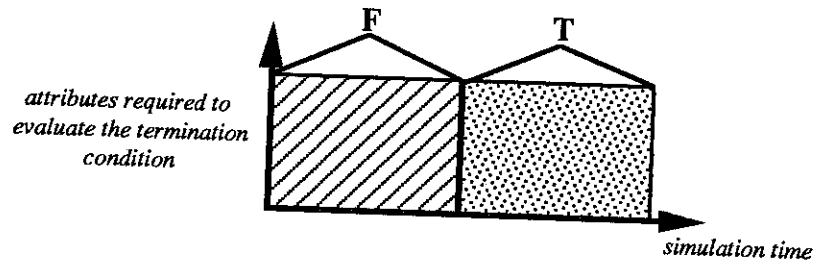

Figure 2: Example of a stable termination condition. "T" and "F" denote regions of time-space in which the condition is *true* and *false*, respectively.

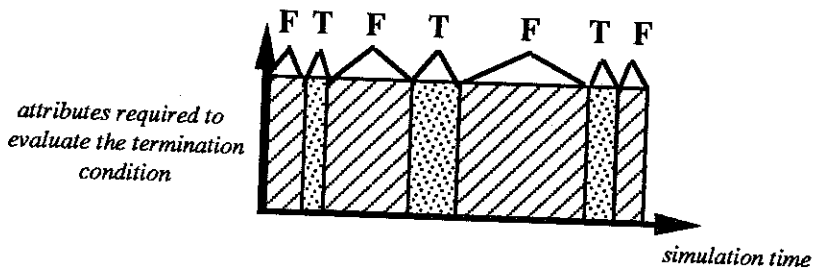

Figure 3: Example of a non-stable termination condition.

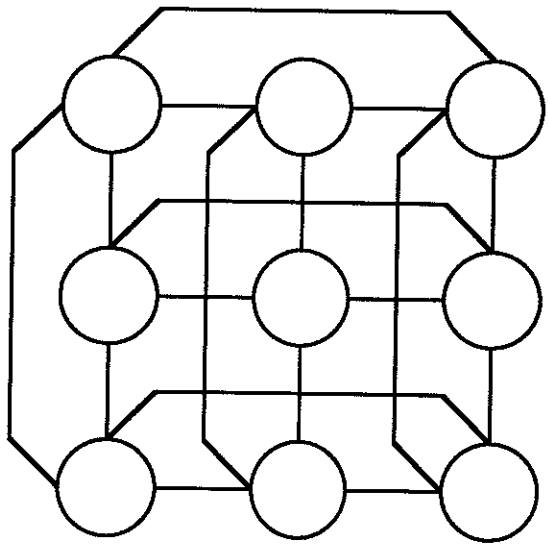

Figure 4: A 3 x 3 torus queuing network

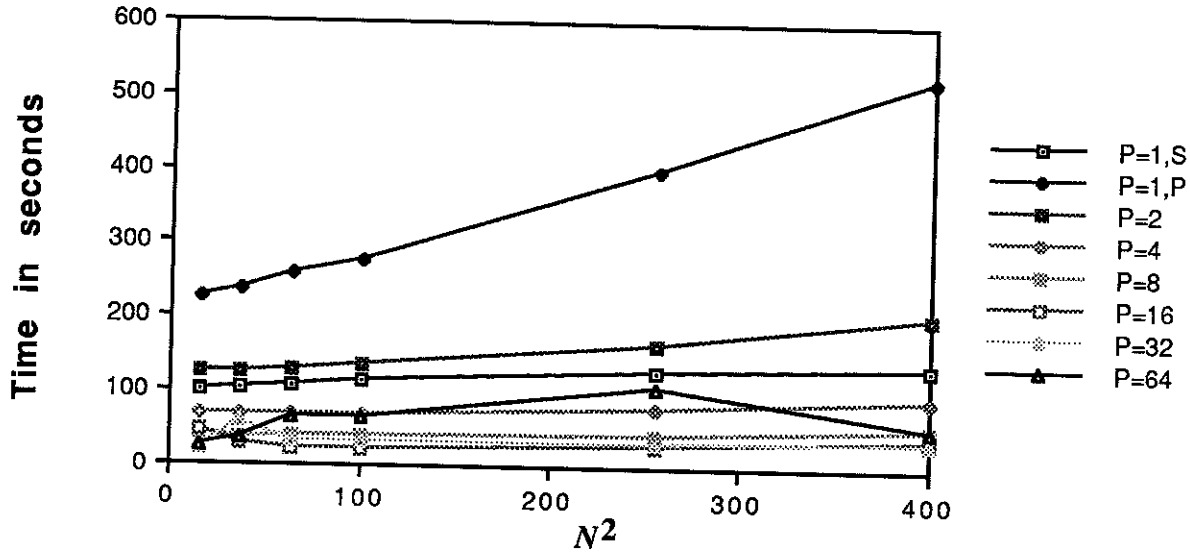

Figure 5: Wall Clock Time required to complete simulation for N^2 *LP*'s with IS + T0.

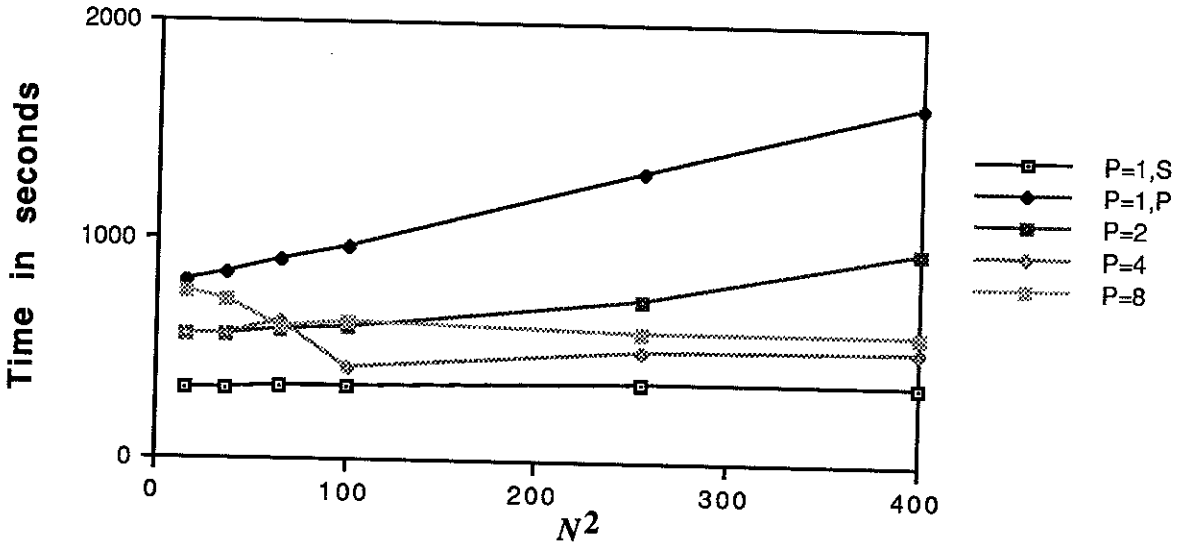

Figure 6: Wall Clock Time required to complete simulation for N^2 *LP*'s with ES + T1.

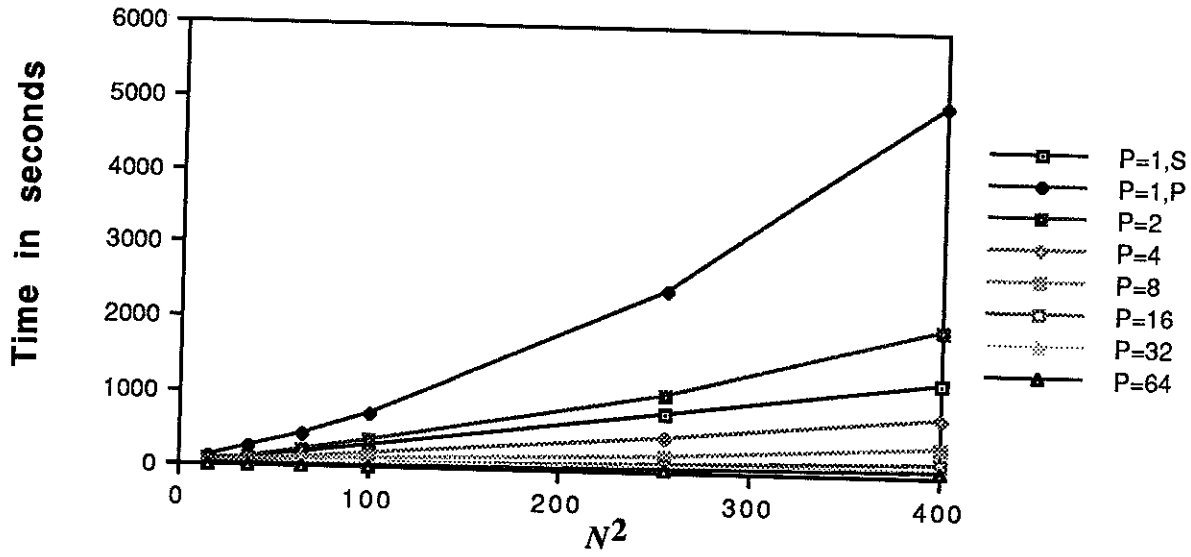

Figure 7: Wall Clock Time required to complete simulation for N^2 *LP*'s with IS + T2.

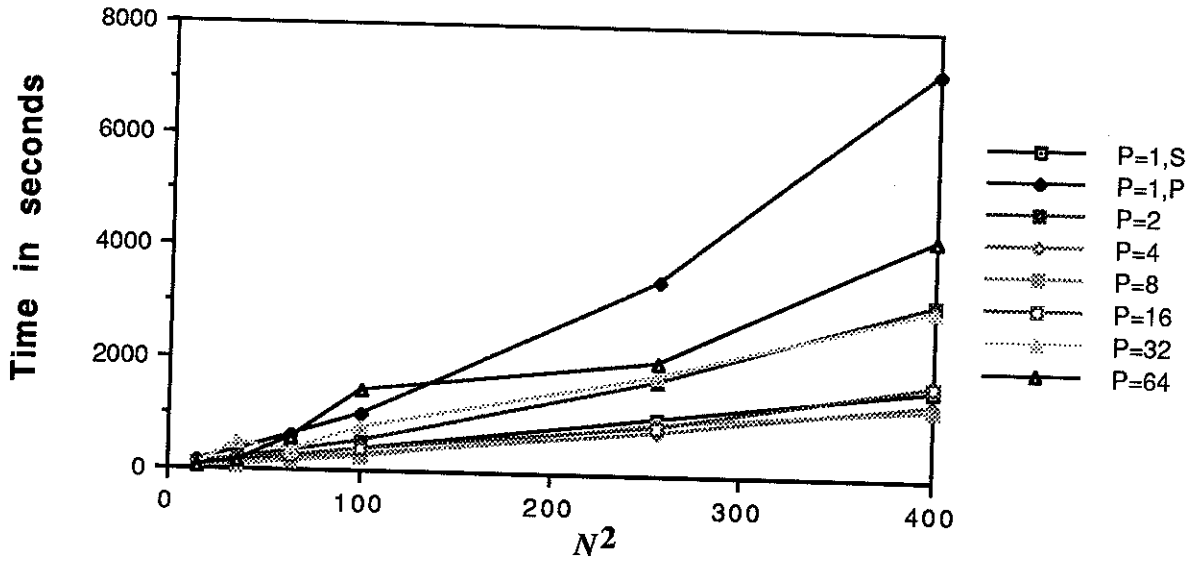

Figure 8: Wall Clock Time required to complete simulation for N^2 *LP*'s with ES + T3.

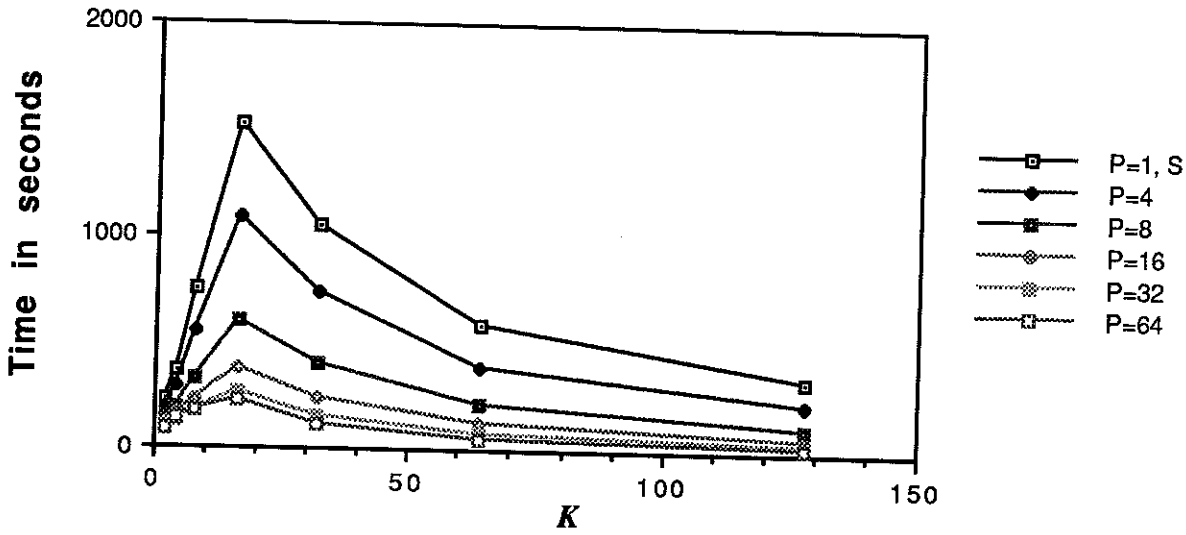

Figure 9: Wall Clock Time required to complete simulation for *K LP*'s with IS + P0, ES + P1.

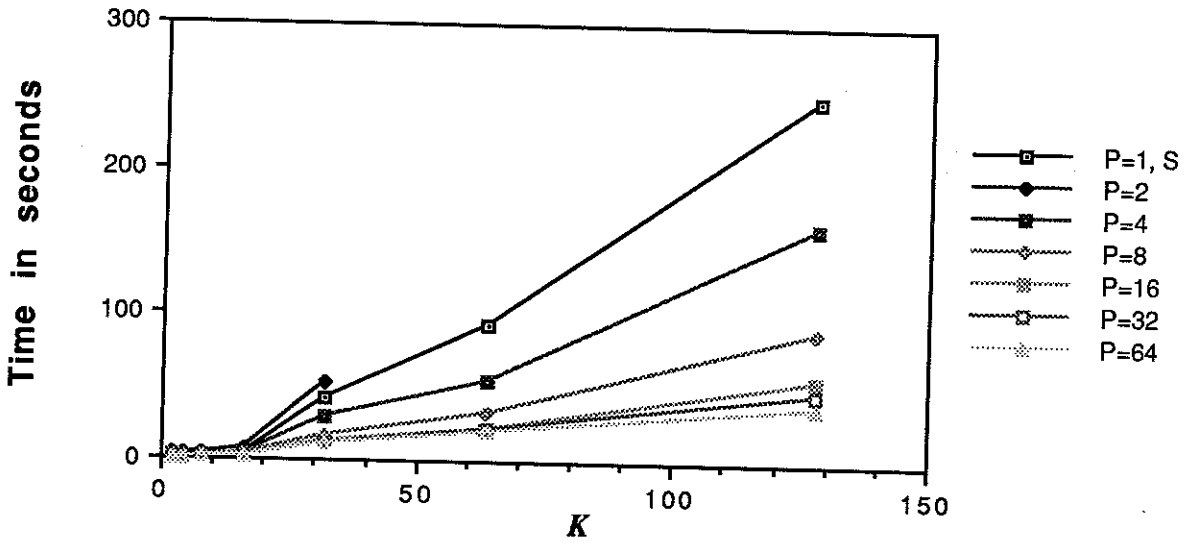

Figure 10: Wall Clock Time required to complete simulation for *K LP*'s with ES + P2.