# Termination and Output Measure Generation in Optimistic and Conservative-Synchronous Parallel Simulations

*Marc Abrams, Vasant Sanjeevan, and Debra S. Richardson*

**TR 92-39**

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

August 3, 1992

# Termination and Output Measure Generation in Optimistic and Conservative-Synchronous Parallel Simulations

Marc Abrams
Vasant Sanjeevan
Debra S. Richardson
abrams@cs.vt.edu

Virginia Polytechnic Institute and State University
Department of Computer Science
Blacksburg, VA 24061-0106

TR 92-39

August 3, 1992

**Abstract**

This paper proposes algorithms to stop parallel discrete event simulations using arbitrary termination conditions and to collect output measures. The *termination problem* for parallel discrete event simulation is to find a value of simulation time, $\tau$, such that a termination condition evaluated using simulation model attribute values at time $\tau$ holds. The output measure *generation problem* is to calculate the value of simulation output measures for the simulation time interval $[0, \tau]$. The paper shows that the time complexity of termination can be higher than that of the underlying simulation; therefore termination can reduce or even preclude speedup. The space complexity of generation is equal to that of the underlying simulation for conservative-synchronous and optimistic protocols; therefore measure generation can increase the space required by parallel simulation by a multiplicative factor. We propose simulation protocol independent algorithms solving the termination and generation problems. Implementations of the algorithms for conservative-synchronous and optimistic protocols are presented. The predicted and measured increase in real time required to execute a time warp and a bounded lag simulation equipped with our termination algorithms is presented. The chief conclusion is that when the time required for each evaluation of the termination condition exceeds the mean time to execute an event, termination can dominate the simulation running time.

# 1 Introduction

Space-parallel discrete-event simulation protocols reported in the literature [8], which execute a simulation model using a set of logical processes (LP's), generally use a single, simple rule to decide when to terminate: each LP terminates when its local simulation clock exceeds a predefined time and the LP will never again receive a message. However, time is only one attribute upon which a termination condition can be based. In addition, parallel simulation protocols generally do not describe how to collect and report output measures.

To solve these problems, this paper proposes two families of termination algorithms, called *Exhaustive* and *Interval*, and three output measure generation algorithms, called *Dissociative*, *Retrospective*, and *Prospective*. The paper also assesses the amount by which these algorithms increase the real time required to execute a simulation when added to optimistic and synchronous-conservative protocols. Optimistic protocols do not explicitly identify the dependencies of events, but save state and roll back whenever an event is scheduled for a simulation time smaller than the current local clock of an LP [9]. Conservative-synchronous protocols identify a *simulation time window* at each LP, which is an interval of simulation time during which the LP can asynchronously execute and ignore events scheduled for it by other LP's [13, 3, 19, 15]. Two benchmarks, described below, are used in the performance assessment, which exemplifies termination conditions and output measures.

**Torus queueing network:** The first benchmark is a simulation of an $N \times N$ toroidal network for any non-preemptive service discipline, where $N$ is an integer [13, Fig. 3]. Each network node is a server with an infinite capacity input queue. A server, modeled by an LP, removes an event, or job, from its input queue and starts service. The service time distribution is bounded below by a non-zero constant. When service completes, the job is inserted in the input queue of one of the four neighbors, selected with equal probability. Initially, servers are idle and have an equal number of jobs in their queues. The following termination conditions are used:

**T0:** Stop when the total number of jobs serviced by all LP's *exceeds* 100,000.

**T1:** Stop when the total number of jobs serviced by all LP's *equals* 100,000.

**T2:** Stop when the local virtual time of each LP exceeds 100,000 simulation time units.

**T3:** Stop when the local virtual time of any LP *first* exceeds 100,000 simulation time units.

Examples of output measures include the number of jobs in each queue when the simulation terminates (TM0), as well as time averages, such as the mean number of customers in the system (TM1).

**Colliding pucks:** The second benchmark is a 50 page C program simulating $K$ pucks moving in two-dimensions on a flat table, colliding with each other and with stationary cushions on the table edge [10]. The pucks remain in constant motion because there is total conservation of energy and no friction. The flat surface is divided into equal sized sectors, which are numbered row wise in ascending order starting from zero. Each sector, puck, and cushion is modeled by an LP. The initial positions and velocities of the pucks are chosen randomly. The following termination conditions are used:

**P0:** Stop when the total number of collisions in all even numbered sectors is *greater than* 500.

**P1:** Stop when the total number of collisions in all even numbered sectors is *equal to* 500.

**P2:** Stop when at least 75% of the pucks lie in even numbered sectors.

Examples of output measures include displaying the final configuration of pucks (PM0), and producing a histogram of the mean time between collisions experienced by all pucks (PM1).

## 1.1 Simulation Termination and Measure Generation Problems

A simulation computes the time evolution of a set of *attributes*. Attributes are partitioned in a mutually exclusive and exhaustive fashion among the LP's in a space parallel simulation; the set of attributes assigned to an LP are *private to* or *owned by* the LP. Each LP is also associated with a *local clock*, containing the simulation time at which the LP last assigned values to its private attributes. The *global clock* of a simulation is the minimum of each local LP clock and, in protocols that use message passing, the timestamp of any message in transit. The global clock is introduced as a convenience for exposition, but is not used in the proposed algorithms.

In the *parallel simulation termination problem,* we are given a non-terminating simulation that generates no output measures, called the *underlying simulation*; a termination condition, denoted $C$; and one or more output measures. Condition $C$ is guaranteed to be *true* when evaluated using the values of simulation model attributes at *some* value of simulation time. This paper proposes and evaluates algorithms to solve the following two problems:

**Termination problem:** Find a value of simulation time, denoted by $\tau$, such that function $C$ evaluated using simulation attribute values at time $\tau$ has value *true* . We say that *termination is detected* when a simulation time has been found that satisfies $C$.

**Generation problem:** Report the value of each simulation output measure for simulation time interval $[0, \tau]$.

The underlying simulation together with algorithms solving the termination and measure generation problems is called an *augmented simulation.*

This problem formulation is based on our belief that in a commercial parallel simulation system, one would like the user to specify a simulation model without worrying about termination (hence the simulation is non-terminating), and then separately specify a variety of termination conditions. The simulation system should automatically superimpose the termination condition on the non-terminating simulation.

We distinguish two types of output measures:

**Class 1:** The output measure is a function of attribute values at multiple simulation times. Usually the measure computes the time average of an attribute, such as measures TM1 and PM1.

**Class 2:** The output measure is a function only of the attribute values at time $\tau$, such as measures TM0 and PM0.

The remainder of this section discusses implications of the termination and generation problems using space-time diagrams, compares the time and space complexity of solving the termination and generation problems to that of the underlying simulation to gauge the problem significance, and discusses why parallel and distributed termination algorithms in the literature

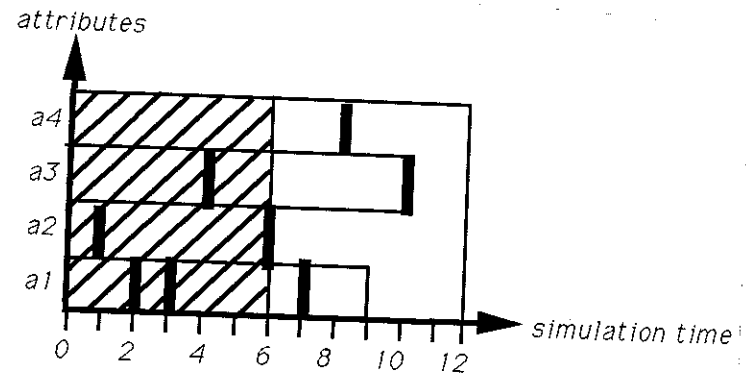



Figure 1: Space-time diagram. Heavy lines indicate assignments of new values to attributes.

are irrelevant to the problems. Section 2 proposes two simulation termination and generation algorithms that can be used with any space-parallel simulation protocol. Sections 4 and 5 present the predicted and measured increases in time required by benchmarks when various termination conditions are used for the time warp and bounded lag protocols. Section 6 discusses another work that treats the termination problem, and Section 7 draws conclusions from the predicted and measured results.

## 1.2 Relation to Space-Time Framework

Chandy and Sherman's space-time diagrams [7] illustrate the inherent complexity of terminating parallel simulation. Chandy and Sherman use space to represent the *set of all LP's* in a simulation program, which presumes the program uses a simulation world view based on processes. To be applicable to any world view [4] (e.g., activity-scan, three-phase approach, event-oriented, as well as process interaction), space in this paper represents *the set of all attributes* in a simulation model.

Figure 1 illustrates a possible snapshot of a simulation with the four attributes $a1, a2, a3$, and $a4$. Each LP has simulated up to a unique simulation time. Let $\Sigma$ denote the set of times at which any attribute is assigned a new value. In Figure 1, $\{1, 2, 3, 4, 6, 7, 8, 9, 10\} \subset \Sigma$, represented by heavy lines. The value of attribute $a3$ remains constant in intervals $[0, 4)$ and $[4, 10)$. The values of simulation model attributes at some time $t$ are represented in the diagram by a vertical line with a constant time coordinate. The termination condition may be evaluated at any simulation time that does not exceed the global clock (which has value 6), indicated by shading.

To terminate a simulation program, termination condition $C(t)$ is evaluated at a sequence of simulation times in $\Sigma$. A sequential simulator typically evaluates the termination condition after each event, object movement, or action occurs, in the event-scheduling, process-interaction, or activity-scan/three-phase-approach, respectively.

Consider next the calculation of output measures. Assume that the space axis in Figure 1 contains the set of attributes required to evaluate the termination conditions. In a sequential simulation, attribute values needed for class 1 measures are accumulated during simulation execution until termination is detected. The space-time manifestation of the sequential simulation is a rectangle bounded by times 0 and $\tau$. If a parallel simulation also accumulates attributes values needed by output measures during simulation, then when the termination condition is detected to hold at time $\tau$, any LP's that have simulated past time $\tau$ have incorrectly accumulated attribute values at simulation times larger than stop time $\tau$. Graphically, the portion of the space-time diagram outside the rectangle bounded by 0 and $\tau$ must be removed from the set of attribute values used to compute class 1 output measures. Class 2 measures are a function of attribute values at a single simulation time, $\tau$. If class 2 measures are computed after termination is detected, then the generation algorithm must provide a means to retrieve attribute values at time $\tau$.

A *stable* termination condition is one that holds forever, once it holds [6]. In the torus and pucks benchmarks, conditions T0, T2, and P0 are stable, while the remaining conditions are unstable. Figure 2 contrasts stable and unstable termination conditions. In (a), there exists a time $t$ such that the termination condition fails for all smaller simulation times and holds for all times equal to or larger than $t$. In (b), the termination condition alternates between *true* and *false* in successive time intervals. The termination problem requires a search of the time axis (e.g., $\Sigma$) for a time satisfying the termination condition. The algorithms proposed here search in ascending time order, however alternate orders, such as using binary search, are proposed in [2].

## 1.3 Complexity Bounds on Termination and Generation

Does the addition of a solution to the termination and generation problems to an underlying simulation potentially increase the real time and space required by the simulation? The *overhead*

a. stable

b. nonstable

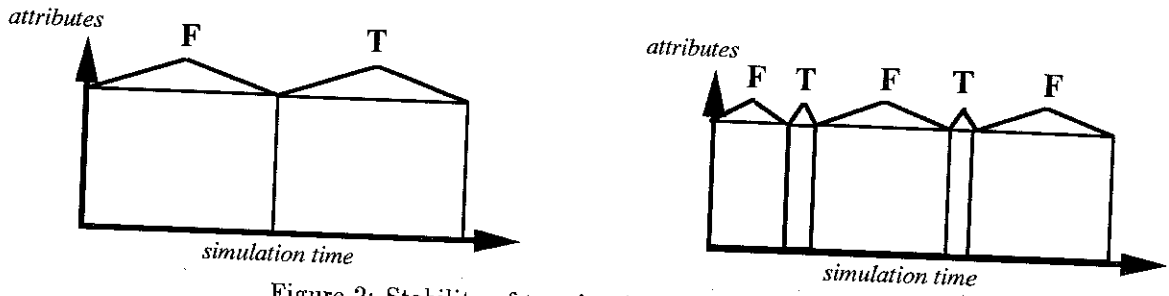



Figure 2: Stability of termination conditions.

*time and space* of termination and generation is the difference between the real time and space required by an augmented simulation and its underlying simulation. The *detection delay* is the difference in real time between when the termination is detected and when the global clock first equals or exceeds $\tau$.

*Constant* overhead time is insignificant and the problem need not be considered further. Overhead with the *same complexity* as the underlying simulation, will be noticeable; for example the running time and space might double or triple. Overhead with *higher complexity* than the complexity of the underlying simulation makes efficient implementation of termination become *more* important than efficient implementation of the underlying parallel simulation. *Unbounded* overhead, if the time and space of the underlying simulation is bounded, can make parallel simulation infeasible. The following arguments, complementing measurements in successive sections, show that in the worst case all four possibilities can arise in parallel simulation protocols.

Let $L$ denote the number of LP's in a simulation. The following assumptions will be used:

**Assumption 1:** *[Holds for entire paper.]* Each evaluation of $C$ or a class 2 measure requires $O(L)$ attributes, and $O(1)$ of these attributes are private to any single LP.

**Assumption 2:** *[Holds for remainder of Section 1.3.]* Each LP assigns new values to its private attributes $O(1)$ times between updates to its local clock.

**Assumption 3:** *[Holds for remainder of Section 1.3.]* For each time in $\Sigma$, $O(1)$ LP's assign new values to their private attributes.

Assumption 1 identifies a computationally intensive class of functions; termination condition P2 fits the assumption. Assumption 2 rules out simulations that contain many zero-time events

in an LP. Assumption 3 rules out simulations that contain many simultaneous events among LP's.

**Time Complexity:** Let $P$ denote the number of processors. For this discussion, we assume that $L$ is an integral multiple of $P$. Let $t_{u,1}$ (respectively, $t_{u,P}$) denote the real time required to execute the underlying simulation with one processor ($P$ processors). Symbols $t_{a,1}$ and $t_{a,P}$ are defined analogously, but represent the real time required by the augmented simulation. Let $t_{o,s,1}$ (respectively, $t_{o,s,P}$) denote the overhead time in the augmented simulation when a stable condition is detected and $C$ is evaluated using 1 ($P$) processor(s). Quantities $t_{o,u,1}$ and $t_{o,u,P}$ are defined analogously for unstable conditions. The augmented simulation real time equals the underlying simulation real time and the overhead time (e.g., $t_{u,P} + t_{o,u,1}$ for $P$ processors executing the underlying simulation and 1 processor executing termination and generation).

Assume that each LP advances its clock $A$ times during simulation. With one processor, the underlying simulation requires time $t_{u,1} = O(LA)$. A lower bound on the parallel execution time is $t_{u,P} \geq t_{u,1}/P$.

To make the analysis apply to shared or distributed memory architectures, the paper will refer to an LP "making the value of an attribute accessible" to another LP or operating system process. In a distributed memory architecture this is done by a message. In a shared memory architecture this could be done by a memory copy, through the use of locks, or possibly requires no action if the memory location storing the attribute is accessible to both LP's and the two are guaranteed not to concurrently access the location.

Consider next a termination algorithm. Let $n_{e,s}$ (respectively, $n_{e,u}$) denote the number of times stable (unstable) condition $C$ is evaluated during simulation. An unstable termination condition must be evaluated once for each time in $\Sigma$; $n_{e,u} = \| \Sigma \| = O(LA)$. Without *a priori* knowledge of any simulation time at which $C$ holds, $n_{e,s}$ must be proportional to the duration of the simulation; hence $n_{e,s} = O(A)$. Let $t_{e,1}$ (respectively, $t_{e,P}$) denote the time required for each evaluation using 1 ($P$) processor(s), equal to the sum of the following two components. (Note that one or more operating system processes will evaluate $C$.) First, the time required to make the attributes required to evaluate $C$ accessible to the process(es) evaluating $C$. A lower

| | Time Complexity | | | | Speedup Bound | |
|---|---|---|---|---|---|---|
| | Underlying Simulation | | Termination | | | |
| | $P=1$ | $P>1$ | $P=1$ | $P>1$ | $P=1$ | $P>1$ |
| Stable | $t_{u,1}=$ $O(LA)$ | $t_{u,P}=$ $O((LA)/P)$ | $t_{o,s,1}=$ $O(LA)$ | $t_{o,s,P}=$ $O(\frac{LA}{P}\log P)$ | $s_{s,1}=$ $O(P)$ | $s_{s,P}=$ $O(P)$ |
| Unstable | $t_{u,1}=$ $O(LA)$ | $t_{u,P}$ $O((LA)/P)$ | $t_{o,s,1}=$ $O(L^2A)$ | $t_{o,s,P}=$ $O(\frac{L^2A}{P}\log P)$ | $s_{u,1}=$ $O(1)$ | $s_{u,P}=$ $O(P/\log P)$ |

Table 1: Time and speedup complexities. Underlying simulation complexity is lower bound. Overhead complexity is upper bound.

bound on the time is $O(1)$. Under Assumption 3, for each time in $\Sigma$, $O(1)$ attribute values must be copied. Therefore each evaluation of $C$ requires time $O(1)$ for copies/moves. Second, the time required for the process(es) to evaluate $C$. From Assumption 1, each evaluation of function $C$ requires reading and combining $O(L)$ attribute values. For one processor this requires time $O(L)$, and for $P$ processors organized as a combining tree [11] this requires time $O(L\log P/P)$. The sum yields $t_{e,1}=O(N)$ and $t_{e,P}=O(L\log P/P)$.

Measure generation always has time complexity equal or smaller to that of the underlying simulation, and hence can be ignored. This holds for class 1 measures because their attribute values are accumulated as the underlying simulation executes. For each class 2 measure, the time required is only $O(L)$ by Assumption 1.

Does termination diminish speedup? We define speedup of the underlying simulation as $t_{u,1}/t_{u,P}$, is $O(P)$. Let $s_{s,1}$ denote the speedup of the augmented simulation with a stable $C$ evaluated by 1 processor, defined as $(t_{u,1}+t_{o,s,1})/(t_{u,P}+t_{o,s,1})$. Symbols $s_{s,P}$, $s_{u,1}$, and $s_{u,P}$ are defined analogously. The values are computed in Table 1.

For stable conditions the underlying simulation has equal or greater time complexity than the termination algorithm; hence termination does not affect speedup. However, for unstable conditions, termination dominates the time complexity of the augmented simulation. With (respectively, without) parallel evaluation of unstable termination conditions, the lower bound speedup diminishes to $(s_{u,P}=O(P/\log P))$ $(s_{u,P}=O(1))$. Therefore addition of a termination algorithm reduces the lower bound speedup by at least a factor of $\log P$ and at worst precludes any speedup with an unstable $C$.

**Space Complexity:** The space overhead of a stable termination condition has a lower bound of zero because the termination time, $\tau$, can always be chosen to exceed the local clock of all LP's; this is discussed in the Prospective Generation method of Section 2.2.

Next consider an unstable condition. Recall from Section 1.2 that the portion of space-time computed that lies outside time interval $[0, \tau]$ must be removed from the set of attributes used to compute class 1 measures. Class 2 measures require evaluation using attribute values at simulation time $\tau$. There are two ways to perform these actions (Let $t_f$ denote the largest simulation time at which the termination condition has been evaluated false.):

1. *Rollback:* After termination is detected, the values of all attributes are reset to their value at simulation time $\tau$. All LP's are restarted, and each LP uses the termination condition, "do not send output messages when my local clock exceeds $\tau$." By this process, class 1 measures are automatically corrected, and class 2 measures are computed using the final values of attributes, which corresponds to simulation time $\tau$. Even non-optimistic protocols can rollback, by simply restoring the initial attribute values, which amounts to executing the simulation twice. The storage required for rollback is $O(L)$.

2. *Save the value of each attribute required to evaluate an output measures at all simulation times that exceed $t_f$:* Class 2 measures are accumulated only for interval $[0, t_f]$. When termination is detected, measures are accumulated for interval $(t_f, \tau]$. Class 1 measures are evaluated when termination is detected, which is straightforward because $\tau > t_f$. Therefore there is no increase in execution time with this method, unlike rollback.

   Using Assumption 3, the storage required is proportional to the *size* of the simulation time interval bounded by $t_f$ and the LP clock with the largest value. In a conservative-synchronous protocol, the interval size is proportional to the window size, which is or can be bounded. For example the interval size is bounded by $O(L)$ in Section 5. Conservative-asynchronous protocols are not considered further because their interval size is unbounded.

What is the space complexity of termination? Each evaluation of $C$ is similar to evaluation of a class 2 output measure. However, rollback is inappropriate because $C$ is computed multiple times, unlike class 2 measures. Hence at any point in the simulation, a termination algorithm

requires at worst the value of each attribute required to evaluate $C$ for all simulation times larger than $t_f$. This value is unbounded for conservative-asynchronous protocols, and can be bounded to any desired value at an increase in running time for optimistic protocols (using the cancel back protocol) and conservative-synchronous protocols (by bounding the window size).

## 1.4 Relation to Classical Termination Algorithms

There are many papers that address the classical parallel program termination problem; Mattern provides an overview of the literature [14]. The classic termination problem differs from the simulation termination problem in three ways. First, in the classic problem, "termination detection" means detecting a stable condition, whereas many interesting simulation termination conditions are unstable. Second, a termination condition must be a conjunct of predicates, each using variables private to one, unique logical process. However, a simulation termination condition generally is a function of simulation attributes, each private to one, unique logical process; the examples given at the start of the paper illustrate this point. Third, the program is idle when the termination condition holds, whereas the simulation termination problem presumes that the simulation is non-terminating.

A problem related to termination detection is stability detection [6, Ch. 9], the object of which is to determine when a stable property holds for an ongoing computation. Similarly, we detect $C$ for a non-terminating simulation. Detecting stable simulation termination conditions is equivalent to the stability detection problem. However stability detection algorithms work for any algorithms, while we exploit special properties of simulation programs. In addition we wish to detect unstable conditions.

The output measure generation problem is not addressed in the literature.

The algorithms presented next are based on our earlier work [1, 17]. Alternate solutions to the termination and generation problems appear in [2, 12].

# 2 Protocol Independent Termination and Generation

## 2.1 Termination Problem Solution

We propose two termination algorithms, which are formally stated and proven correct by Richardson [16]:

**Exhaustive Termination:** Evaluate $C(t)$ for each $t \in \Sigma$ in ascending order until $C(t) = true$ .

**Interval Termination:** Choose $\Sigma_S \subset \Sigma$ such that $\Sigma_S$ contains a time for which $C(t) = true$ . Evaluate $C(t)$ for each $t \in \Sigma$ in ascending order until $C(t) = true$ .

Exhaustive termination works for any termination condition. Interval termination requires the ability to select a set $\Sigma_S$. This subset is easy to select for a stable termination condition. One example is to include in $\Sigma_S$ every $n^{\text{th}}$ value of $\Sigma$ in ascending order. The optimal interval is problem dependent. Too small an interval may increase the real simulation time because the condition is evaluated too frequently. Too large an interval can increase the detection delay and hence the real simulation time.

The time complexity of Exhaustive (respectively, Interval) Termination is that listed for unstable (stable) conditions in Table 1. The space complexity can be bounded to any desired value as discussed in Section 1.3.

### 2.1.1 Combinatorial Termination Conditions

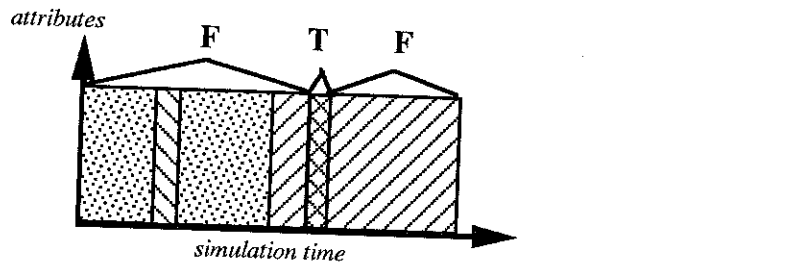
In this section we consider termination conditions that are a conjunct or disjunct of two terms, denoted $C_1$ and $C_2$, where each term is either a stable or unstable condition. It is straightforward to generalize the discussion to more than two terms. Table 2 summarizes the discussion.

**Conjunct of Stable Conditions:** The conjunct is stable condition; hence use Interval Termination.

**Conjunct of Unstable Conditions:** The conjunct is usually unstable; thus use Exhaustive Termination. However the conjunct may be stable (e.g., $C_1$ holds in simulation time interval $[4, 6)$ and $C_2$ holds for all simulation times larger than 5), permitting Interval Termination.

| *Condition Form* | *Termination Category* | *Algorithm* |
|---|---|---|
| Conjunct | all stable terms | Interval |
| | all unstable terms | Conjunct stable: Interval<br>Conjunct unstable: Exhaustive |
| | at least one stable, one unstable condition | Interval_then_Exhaustive |
| Disjunct | all stable terms | Interval |
| | all unstable terms | Disjunct stable: Interval<br>Disjunct unstable: Exhaustive |
| | at least one stable, one unstable condition | Either Interval, Exhaustive, or both in parallel |

Table 2: Termination algorithms for various termination condition forms.




KEY for c:

:both conditions are false

:nonstable true, stable false

:stable true, nonstable false

:both conditions are true

Figure 3: The conjunction of the stable and unstable condition shown in Figure 2.

**Conjunct of Stable, Unstable Conditions** (Figure 3): Exhaustive Termination may be used, but is inefficient because any exhaustive search before the stable condition holds is useless. We propose *Interval_then_Exhaustive Termination*, appropriate for optimistic and any protocol that can be modified to include roll back, such as conservative-synchronous. Execute Interval Termination until a time is found where the stable term holds. Then roll back to the last time at which the stable term evaluated to *false* . Then apply Exhaustive Termination.

**Disjunct of Stable Conditions:** The disjunct is stable; hence use Interval Termination.

**Disjunct of Unstable Conditions:** The disjunct is usually unstable, requiring Exhaustive Termination. However the disjunct may be stable, permitting Interval Termination.

**Disjunct of Stable, Unstable Conditions:** The disjunct is unstable, but it can be detected by either Interval, Exhaustive, or both in parallel. Neither algorithm is guaranteed to minimize detection delay. For example, assume that $\Sigma$ is the set of nonnegative integers, that $C_1$ holds in $[2, 20]$, and $C_2$ holds for all times larger than $10^6$. Exhaustive Termination has a smaller detection delay than Interval Termination.

## 2.2 Generation Problem Solution

We propose below three solutions which are independent of the termination solution (e.g., Exhaustive, Interval Termination). *Dissociative Generation* decouples the simulation from calculation of output measures. *Retrospective Generation* selects $\tau$ in the past of all LP's (e.g., each LP's local clock exceeds $\tau$), while *Prospective Generation* selects $\tau$ in the future of all LP's. All three algorithms require the addition of one operating system thread or process, referred to as MG (measure generator).

**Dissociative Generation:** (Suitable for protocols without rollback, such as conservative) Each time that a new value is assigned to an attribute required for evaluation of any output measure, the LP that owns the attribute makes accessible to MG the simulation time at which the assignment occurs along with the new value. The termination algorithm communicates to MG advances to $t_f$. MG accumulates class 1 measures for simulation times less than $t_f$ as it receives new attribute values. When termination is detected, MG evaluates the class 2 measure using attribute values at time $\tau$.

**Retrospective Generation:** (Suitable for optimistic and any protocol that can be modified to include rollback, such as conservative-synchronous) LP's accumulate class 1 measures as they simulate. When termination is detected, $\tau$ is distributed to all LP's, and each LP is rolled back to simulation time $\tau$. The rollback automatically corrects the value of class 1 measures. Each LP that owns an attributed required for a class 2 measure makes the current attribute values, corresponding to time $\tau$, accessible to MG and terminates. MG calculates the measure when it has received all required attributes.

**Prospective Generation:** (Suitable for any protocol and all stable conditions, and some unstable conditions) LP's accumulate class 1 measures as they simulate. When a simulation time $t$ is found at which $C$ holds, LP's are inhibited from simulating. An upper bound on the maximum of the local clock value of each LP and the send time of each in-transit message is calculated, denoted $T_H$.

Based on the value of $t$, the Termination Detector selects a time $\tau > T_H$, such that $C(\tau)$ is guaranteed to be true. The algorithm broadcasts the value $\tau$ to all LP's, and the LP's are permitted to continue simulation. Each LP continues simulation until its local clock equals or first exceeds $\tau$. Class 1 output measures are correct at this point. Class 2 measures are calculated as described for Retrospective Generation.

It is always possible to select a $\tau > T_H$ at which $C(\tau)$ holds for stable termination conditions: by the definition of stability, $C$ is *true* for any $\tau$ that equals or exceeds $T_H$. It is sometimes possible to select a $\tau > T_H$ for certain unstable termination conditions using Predictive Termination [2].

# 3 Experiment Design

The performance assessment in sections 4 and 5 uses the metric of real time required to complete a simulation, denoted $R$.

In the torus network from Section 1, each server initially has 200 jobs in its input queue. Experiments with the extreme values of $N$ ($N = 3$ and $N = 20$) for $P = 1, 2, 4$, and 8 with termination condition T1 using termination algorithm ES23 from Section 5.2 show that 100 jobs is sufficient so that no server is ever idle during simulation. Therefore $R$ is independent of the constant 200.

We use an exponentially distributed service time with a mean, $x$, of 200 simulation time units and a constant, $y$, of 100 time units added to it. Metric $R$ changes by less than 1% in three experiments on termination algorithm ES23 and termination condition T1 in Section 5 using $N = 3, 20$; $P = 1, 2, 4$, and 8; and the three additional combinations $x = 500, y = 100$; $x = 500, y = 250$; and $x = y = 500$. Therefore $R$ is insensitive to the value of $x$ and $y$.

The data value used to plot each point on the graphs is the mean of the run times of three experiments.

Our experiments contain a termination condition, but no output measures, because generation does not increase the time complexity of the underlying simulation as discussed in Section 1.3.

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 $P$.

# 4 Optimistic Protocol Termination and Generation

Because time warp is based on a message passing model, so will our discussion in this section. Time warp terminates as follows: Each LP sets its simulation clock to time positive infinity when its input queue is empty. The simulation stops execution when global virtual time (GVT) advances to positive infinity. GVT is a lower bound on the value of all LP's simulation clocks and the sending timestamp of all messages in transit or in any LP's input queue [9].

We propose four algorithms (ES, IS, EP, and IP) that incorporate Exhaustive and Interval Termination along with Retrospective Generation into time warp. The algorithms are implemented solely as part of the user's simulation, and can therefore be added to any existing time warp simulation without modification to the time warp protocol itself.

We propose the addition of two new LP's: TD (*Termination Detector*) and MG; the other LP's in a simulation are called *underlying* LP's. Before each underlying LP starts simulating, it sends TD the initial value of each such attribute in an UPDATE message. Later, whenever an underlying LP changes the value of an attribute required to evaluate termination condition $C$, the LP sends the new value to TD in an UPDATE message. Whenever TD receives an UPDATE message, it evaluates $C$ at the simulation time contained in the send timestamp of the UPDATE message.

**Assumption 4:** *[Holds for remainder of paper]* A simulation program computes a single class 2 measure, denoted $F$. Each LP owns exactly one attribute required to evaluate functions $C$ and $F$.

Assumption 4 simplifies the presentation, but is straightforward to relax.

Figures 4-5 contain the algorithm used as the basis for ES, IS, EP, and IP. (The algorithms in this paper use the C language syntax.) When TD finds a simulation time $\tau$ at which $C$ is true, TD broadcasts a TERMINATE message to all underlying LP's with a send time of $\tau$, which causes them to cease simulation of events. Whenever $\tau$ is smaller than the local clock of the receiving LP, the LP will roll back to $\tau$.

Because TD appears to time warp as another LP, TD is subject to roll back whenever it receives an or UPDATE message with send time smaller than its local clock value. Therefore TD may detect $C$ as true, but then receive attribute values with send times smaller than its local clock, and roll back. Because TD is another time warp LP, any TERMINATE messages that TD sends may be canceled upon rollback.

Measure generation is done as described earlier for Retrospective Generation, noting that MG computes the class 2 measure when it receives a TERMINATE message.

We categorize termination algorithms for optimistic protocols based on whether termination detection is done using the Exhaustive Termination (category E) or Interval Termination (I), whether the evaluation of parallelized and done by more than one processor (P). Termination algorithms are denoted by two letter mnemonics, such as ES.

## 4.1 Algorithm Performance

We give a prediction followed by measurement of $R$. Each case is identified by a termination algorithm and a rule, such as IS+T0 for Interval Termination with termination condition T0.

Experiments are only performed with algorithms IS and ES, because the maximum simulation size feasible with the time warp implementation used in Section 4.1.2 is not large enough for the time required to evaluate $C$ to dominate $R$.

### 4.1.1 Predicted Performance

**IS+T0:** Each LP sends one UPDATE message to TD each time it processes $N^2/10$ messages. The value $N^2/10$ is selected to make the arrival rate of messages at TD is independent of $N$ and to minimize termination delay. (The sensitivity of $R$ to the choice of constant 10 is explored in

```
#define Update       0
#define Terminate    1
#define MeasureValue 3
typedef Time float;

struct Attribute {
  unsigned int Index;
  float Value;
};

struct Message {
  unsigned int Type;              /*Update or Terminate                  */
  Time SendTime;
  Attribute Attr;
};

function Receive(Message* M);     /*Remove next message from input queue*/
function Broadcast(Message* M);   /*Send M to all underlying LP's        */
function Send(Message* M, LP* L) /*Send M to single destination: LP L   */
function C(t);                    /*Termination condition                */

function TD_Exhaustive() {
  float A[1..L];
  Message* M;
  int i;
  Time StopTime=0;

  /*Record initial attribute values.*/
  for (i=0; i<N; i++) {
    Receive(M);
    /*M->Type == InitialValue*/
    A[M->Attr->Index] = M->Attr->Value;
  }

  /*Evaluate C each time a new attribute value arrives*/
  while (StopTime==0) {
    Receive(M);
    /*M->Type == Update*/
    A[M->Attr->Index] = M->Attr->Value;
    if (C(M->SendTime)) {
      StopTime=M->SendTime;
      M->Type=Terminate; M->SendTime=StopTime;
      Broadcast(M);
      }
  }
}
```

Figure 4: Basis of algorithms ES, IS, EP, and IP for time warp (part 1 of 2)

```
    function underlying_LP(int i) {    /*i is LP's id*/
      Time StopTime=0;
      Message* M = malloc(sizeof(Message));
      M->Type=InitialValue;

      /*Send initial value of each attribute required for termination to TD*/
      M->Attr->Index=i; M->Attr->Value=...; Send(M,TD);

      while (StopTime==0) {
        ... /*LP simulation code, unrelated to termination.*/
        Receive(M);
        if (M->Type==Terminate) StopTime=M->SendTime;
        ... /*LP simulation code, unrelated to termination.*/
      }

      /*Send current value of each attribute required for output measure to MG*/
      /*For class 2 measures only!*/
      M->Type=MeasureValue; M->Attr->Index=i; M->Attr->Value=...; Send(M,MG);

      /*Receive and discard all incoming messages, because their send time
        exceeds StopTime.*/
      while (1) Receive(M);
    }

    function MG() {
      float A[1..L];
      Message* M;
      int i;

      /*Receive attribute values required to evaluate function F*/
      for (i=0; i<N; i++) {
        Receive(M);
        A[M->Attr->Index] = M->Attr->Value;
      }

      output F(M->SendTime,A);
    }
```

Figure 5: Basis of algorithms ES, IS, EP, and IP for time warp (part 2 of 2)

Section 4.1.2.)

As $N$ increases, $\tau$ decreases because the number of servers processing jobs increases. Therefore, we expect a flat curve for $R$ as a function of $N$ for a fixed $P$. Also, for a fixed $N$, we expect a decrease in $R$ as $P$ increases.

**ES+T1:** Each LP sends one UPDATE message to TD after every job it processes so that TD can detect the smallest simulation time at which the 100,000th completes service. Therefore the arrival rate of messages at TD grows linearly with the number of servers, $N^2$. This creates a bottleneck at TD and should increase $R$ for ES+T1 over $R$ for IS+T0. As is the case with T0, the simulation time at which T1 holds decreases as $N$ increases. Therefore, as $N$ increases, each server sends TD fewer messages, decreasing $R$. Of the previous two effects, the tendency of TD to be a bottleneck should dominant, and we expect a linear curve with a small positive slope for $R$ as $N$ increases for a fixed $P$. As before, for a fixed $N$, we also expect a reduction in the running time as $P$ increases.

**IS+T2:** The simulation time at which T2 holds is independent of $N$. Unlike T0, T1, and T3, only *one* LP need send UPDATE messages to TD every time its local clock changes. Therefore the arrival rate of messages at TD is independent of $N$ and thus message processing at TD is not a bottleneck. There is a potential (but low probability) problem. Suppose the LP that sends UPDATE messages assign a large service time (e.g., $10^5$) to the job which last enters service before the LP's local clock is assigned a value satisfying T2 (e.g., time 99,999). This case can increase $R$ by an arbitrarily large amount (doubling $R$ for the values given here). As $N$ increases for a fixed $P$, we expect $R$ to increase linearly and thus get a curve with a small positive slope. Again, for a fixed $N$, we expect a reduction in $R$ as $P$ increases.

**ES+T3:** As with T2, the simulation time at which T3 holds is independent of $N$. To detect T3, every LP sends an UPDATE message to TD every time its local clock changes because we need to detect the first time that the local clock of any LP exceeds 100,000 simulation time units. As with ES+T1, message processing at TD is a bottleneck. As $N$ is increased for a fixed $P$, we expect $R$ to increase either linearly (if TD is not a bottleneck) with a large slope or quadratically

(if TD is a bottleneck). As before, for a fixed $N$, we expect a reduction in $R$ as $P$ increases.

**IS+P0:** The LP modeling each puck sends UPDATE messages to TD periodically. Each LP modeling a puck sends an UPDATE message containing the number of collisions the puck has so far experienced to TD after every 10 collisions. (The sensitivity of $R$ to the constant 10 is explored in Section 4.1.2.) Since both pucks involved in a collision report a collision to TD, TD has to divide its collision count by a factor of 2 to get the correct collision count. As $K$ increases, $\tau$ decreases because the number of collisions per unit simulation time is proportional to $K$. Therefore, we expect to have a flat curve for $R$ as a function of $K$ for a fixed $P$. Also, for a fixed $K$, we expect a decrease in $R$ as $P$ increases.

**ES+P1:** The LP modeling each puck sends an UPDATE messages to TD after *every* collision. As in IS+P0, $\tau$ decreases as $K$ increases. However, the arrival rate of messages at TD might be high enough for TD to become a bottleneck. Therefore, we expect to have either a flat curve or a curve with a small positive slope for $R$ as a function of $K$ for a fixed $P$. Again, for a fixed $K$, we expect a decrease in $R$ as $P$ increases.

**ES+P2:** The LP's modeling even numbered sectors send UPDATE messages to TD every time a puck enters or leaves their sectors. The TD accordingly increments or decrements its count of pucks in even numbered sectors. Time $\tau$ is independent of $K$. Therefore, we expect to have a curve with a positive slope for $R$ as a function of $K$ for a fixed $P$. For a fixed $K$, we would expect a decrease in $R$ as $P$ is increased.

### 4.1.2 Measured Performance

Our simulations are implemented on a BBN Butterfly with 84 processors using JPL's Time Warp Operating System (TWOS) Version 2.7.1 (Jefferson 1987). 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 values of $P$ used for the parallel simulation is varied from 1 to 64 through powers of 2. We also run each combination on JPL's TWSIM sequential simulator.

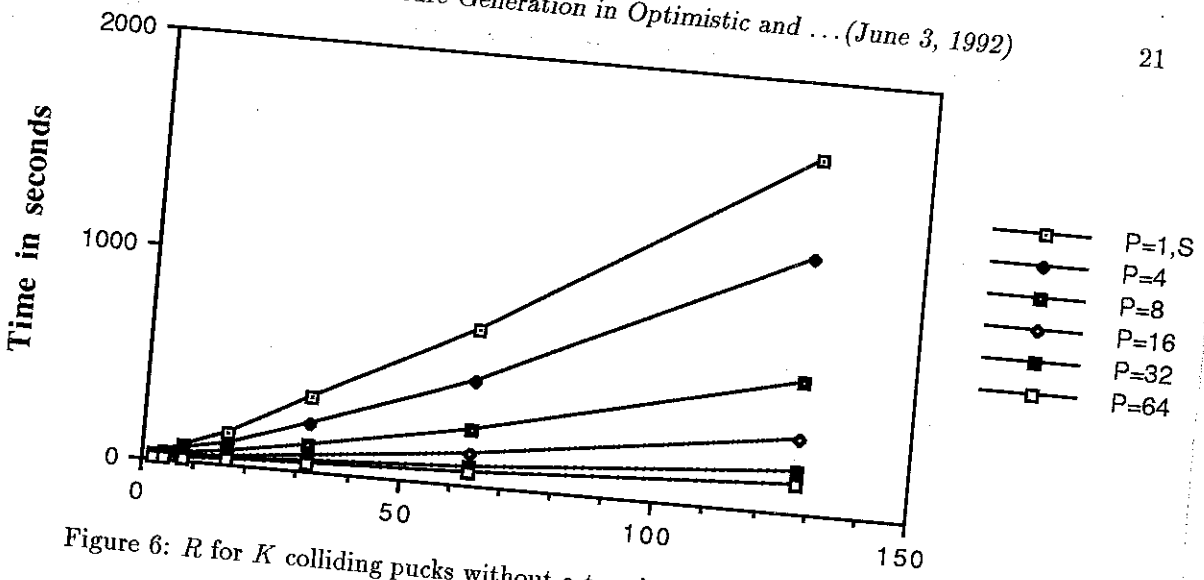



Figure 6: $R$ for $K$ colliding pucks without a termination algorithm under time warp.

The simulations are statically load balanced for all runs using the TWOS load balancer prior to measurement. The load balancer calculates the assignment of LP's to physical processors that minimizes $R$. 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 is done in a similar manner.

TWOS allows a user to specify a simulation time at which all LP's should be terminated. Using a termination time of 100,000 (equivalent to T2), the torus benchmark achieves a 2-3% smaller value of $R$ than the case of IS+T2, discussed below. Metric $R$ for the pucks simulation, with the same termination time, is illustrated in Figure 6.

**IS+T0 (Figure 7):** The curves have a small positive slope, which we conjecture is due to an increase in message passing overhead as $N$ increases. In the parallel cases, $R$ decreases as $P$ decreases from 1 to 32, but it increases with 64 processors. An LP is in *saturation* when the mean interarrival time of messages to the LP is equal to the mean time required to process each message. We conjecture that with 64 processors, TD is close to saturation. When fewer than 4 processors are used, $R$ for the parallel simulation exceeds $R$ for the sequential simulation.

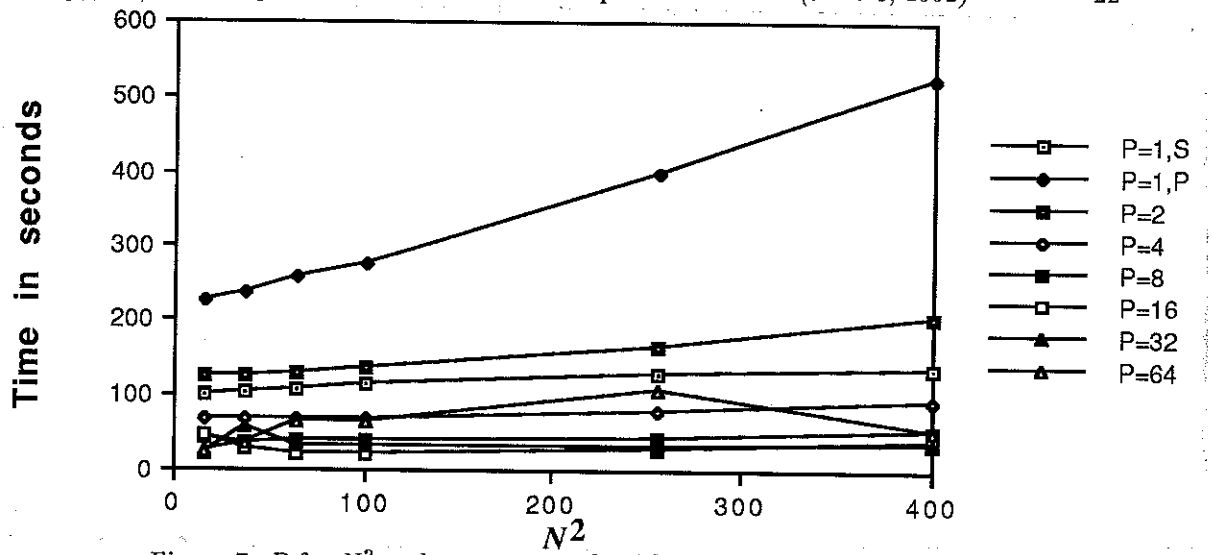



Figure 7: $R$ for $N^2$ node torus network with IS+T0 under time warp.

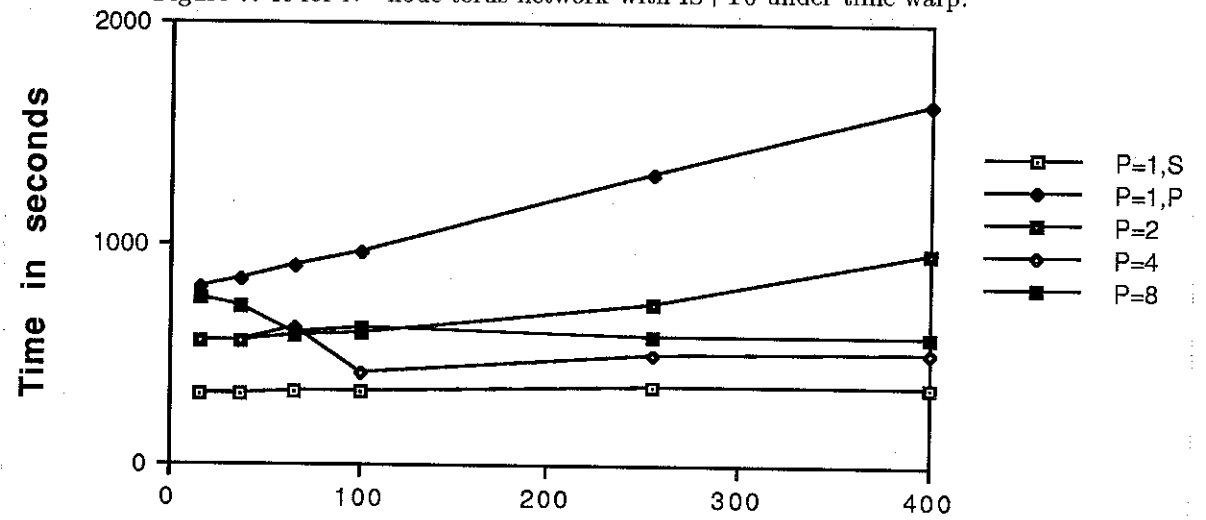



Figure 8: $R$ for $N^2$ node torus network with ES+T1 under time warp.

$R$ is sensitive to the rate with which UPDATE messages are sent by the LP's (e.g., $N^2/10$). However, changing the rate to $N^2/5$ or $N^2/20$ increases $R$. Sending UPDATE messages with rate of $N^2/5$ increased $R$ by 5% for $K = 2$ and $P = 1, 2, 4, 8, 16$ and 32, and by 10% for $K = 128$ and $P = 1, 2, 4, 8, 16$ and 32. Sending UPDATE messages with rate of $N^2/20$ increased $R$ by 10% for $K = 2$ and $P = 1, 2, 4, 8, 16$ and 32, and 15% for $K = 128$ and $P = 1, 2, 4, 8, 16$ and 32.

**ES+T1 (Figure 8):** The curves have a positive slope, as predicted. As $P$ increases from 1 to 8, $R$ decreases, but $R$ grows disproportionately and erraticly for $P > 8$. As in the IS+T0 case, we conjecture that TD is close to saturation at this point.

For all values of $P$, $R$ for parallel simulation always exceeds $R$ for a sequential simulation.

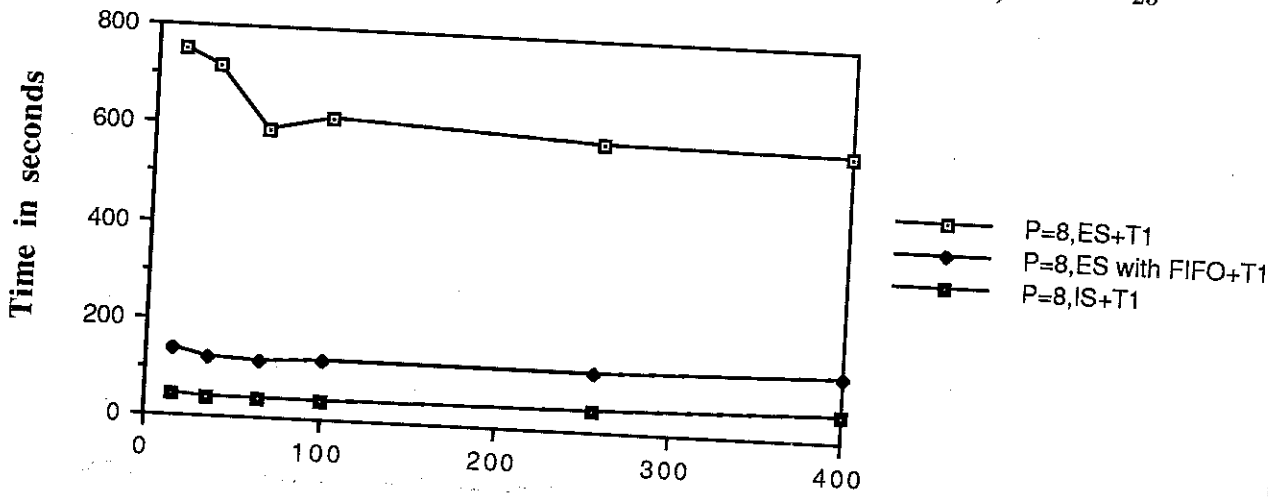



Figure 9: $R$ for $N^2$ node torus network comparing ES+T1, ES+T1 using FIFO queues, and IS+T1 under time warp.

Therefore changing a single word in the termination condition – "exceeds" versus "equals" – requires switching from an interval termination algorithm (IS+T0) to an exhaustive termination algorithm (ES+T1) for the torus network benchmark and *precludes speedup.*

To reduce the termination overhead, we reimplemented ES as follows. Each LP is given one FIFO queue for each attribute that is required to evaluate $C$. The FIFO queues are initially empty. Rather than sending one UPDATE message each time an attribute required to evaluate $C$ is assigned a new value, the LP stores the LP's current local clock and the new attribute value into the FIFO queue. When the queue is fully, its contents are sent to TD, and the queue is emptied. The resulting implementation improves the performance of ES+T0 to within a small constant of IS+T0 for the $P = 8$ case using a queue size of 20 (Figure 9).

**IS+T2 (Figure 10):** The curves have a positive slope, as predicted. As $P$ increases from 1 to 64, $R$ decreases. When fewer than 4 processors are used, $R$ is larger for parallel simulation than for sequential simulation.

**ES+T3 (Figure 11):** The curves have a large slope, as predicted. The running time decreases as $P$ increases from 1 to 16, but increases with 32 and 64 processors. This could again be due to TD nearing saturation. Once again the sequential simulation does better than any of the parallel simulations.

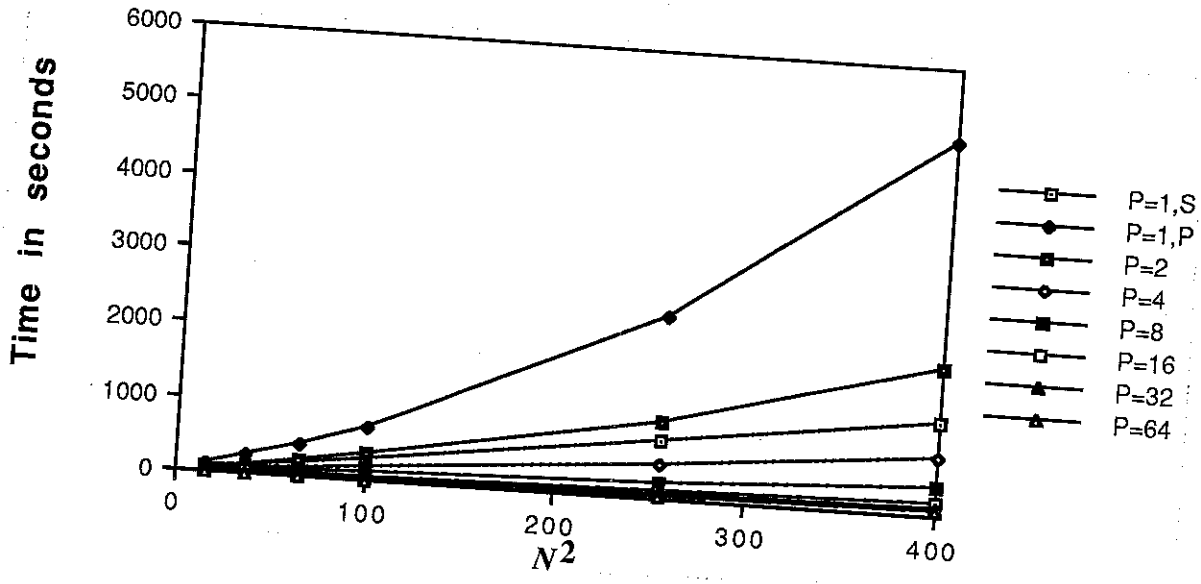



Figure 10: $R$ for $N^2$ node torus network with IS+T2 under time warp.

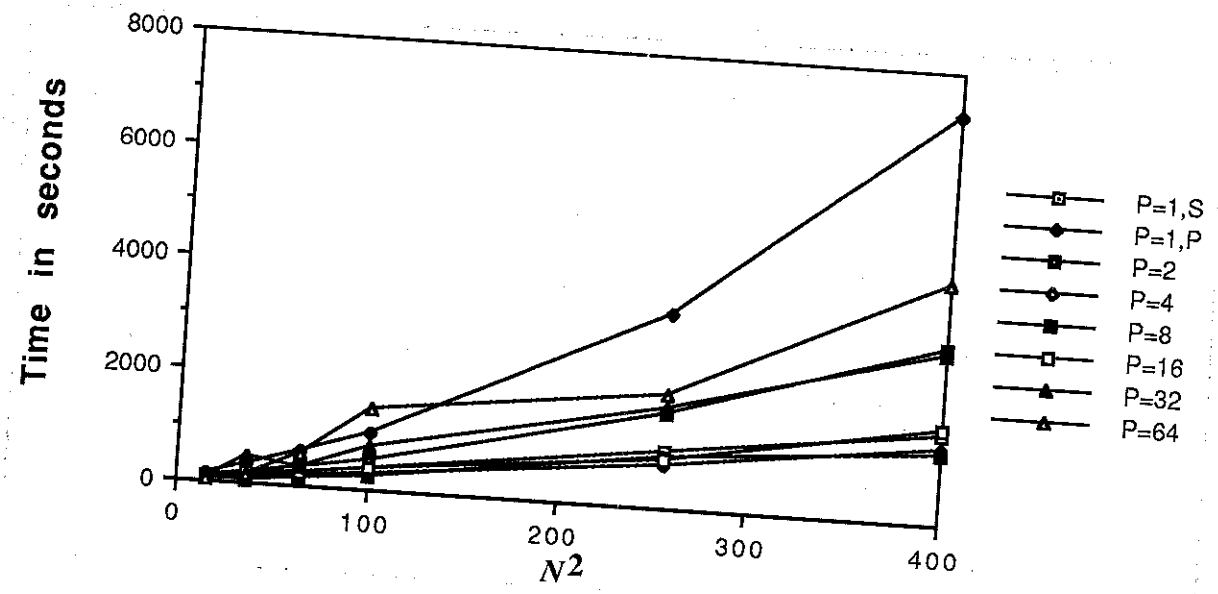



Figure 11: $R$ for $N^2$ node torus network with ES+T3 under time warp.

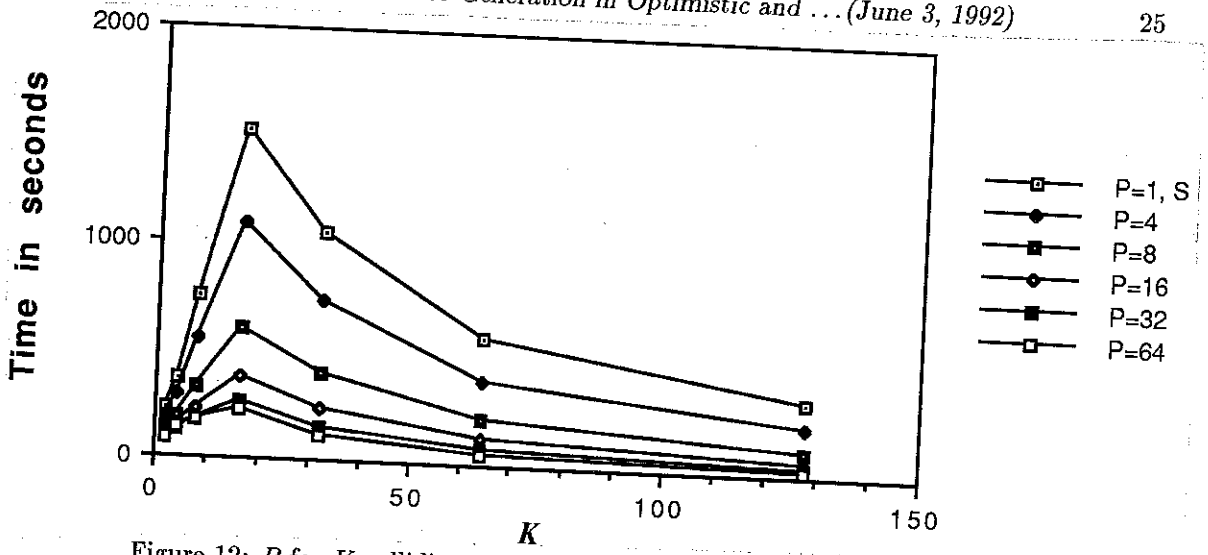



Figure 12: $R$ for $K$ colliding pucks with IS+P0 and ES+P1 under time warp.

**IS+P0 (Figure 12):** The curves have a hump at $K = 16$ but are otherwise essentially flat. As $P$ increases from 4 to 64, $R$ for the parallel cases decreases. When more than 4 processors are used, $R$ for parallel simulation exceeds $R$ for sequential simulation. The simulation runs out of memory for $P = 1$ and P=2. We investigated the sensitivity of $R$ to UPDATE frequency, varying it from the value of 10 used in Figure 12 to 5 and 20 events for $P = 4, 8, 16, 32$ and 64, and $K = 2, 16$ and 128. Metric $R$ changed by less than 2%; therefore $R$ is insensitive to the UPDATE rate.

**ES+P1 (Figure 12):** Measure $R$ for this algorithm is essentially the same as 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 TD is negligible compared to the rate with which messages are generated and processed by the simulation itself.

**ES+P2 (Figure 13):** The curves have a positive slope, as predicted. As $P$ increases from 4 to 32, $R$ decreases. With more than 4 processors, $R$ for parallel simulation exceeds $R$ for sequential simulation. The simulation runs out of memory for $P = 1, 2$.

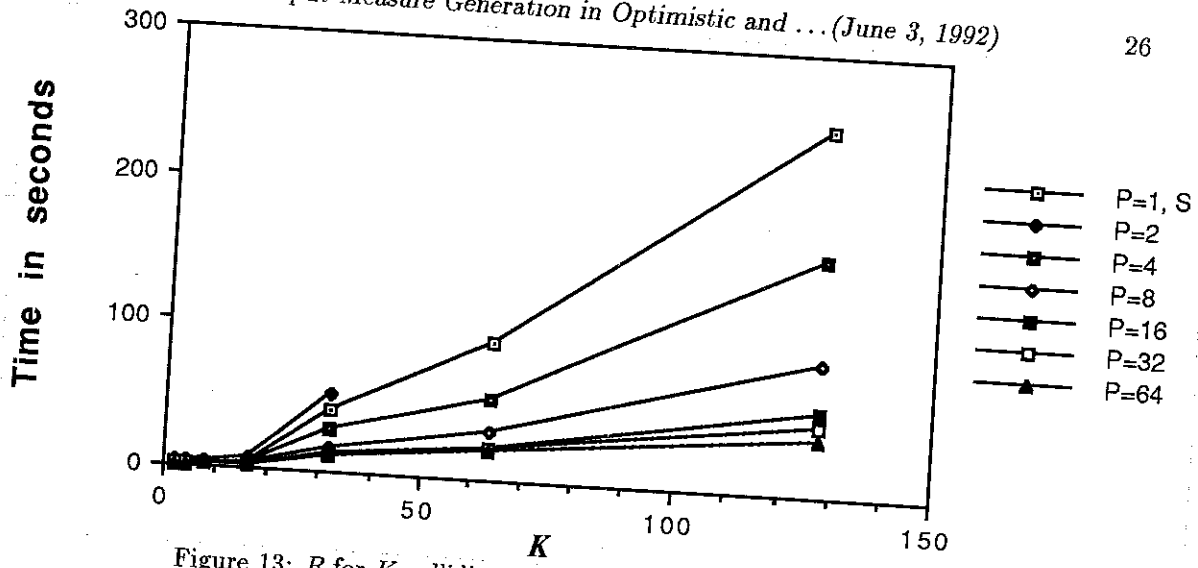



Figure 13: $R$ for $K$ colliding pucks with ES+P2 under time warp.

# 5 Conservative-Synchronous Protocol Termination and Generation

This section proposes implementations of Exhaustive Termination and Interval Termination with Retrospective and Prospective Termination Generation for Lubachevsy's bounded lag algorithm, shown in Figure 14. The implementations discussed here are representative of how to implement termination and generation in other conservative-synchronous protocols. We select bounded lag because its bound on window size also bounds the storage required for termination and generation.

## 5.1 Bounded Lag Algorithm

LP's will be denoted $LP_0, LP_1, \ldots, LP_{M-1}$; let $0 \leq i < M$. All LP's comprising the underlying simulation asynchronously executes in parallel the algorithm of Figure 14. Each $LP_i$ maintains an input queue of events sorted in ascending timestamp order, pointed to by variable `InputQ[i]`. The timestamp of an event defines the simulation time at which an LP will process the event. An LP is said to *schedule* an event on a destination LP when it inserts an event in the event queue of the destination LP.

Each iteration of the outer while loop in Figure 14 contains three sections, separated by barrier synchronizations (denoted "`synchronize`"). First, for all $i$, $LP_i$ computes in `Alpha[i]` a lower bound on the simulation time at which another LP can insert (or schedule) an event

in its input queue. Second, for all $i$, $LP_i$ executes all events in the window with lower bound `Floor` and upper edge equal to the minimum of `Alpha[i]` and a fixed constant, `B`, known as the *bounded lag*. Third, $LP_1$ alone computes the new `Floor`, which is the simulation time of the next event in the entire simulation model. Stop variable `SV` is set to `true` when the termination condition becomes true.

For convenience we assume that each $LP_i$ has at least two attributes local to it, one required to evaluate $C$, stored in `S[i]`, and the other required to evaluate OMF, stored in `O[i]`. Relaxing this assumption is straightforward.

Section 5.2 presents termination algorithms for the bounded lag protocol. Section 5.3 contains predictions followed by measurements of $R$.

## 5.2 Termination and Generation Algorithms

Implementation of Exhaustive and Interval Termination in bounded lag presents several options, explained below. The algorithms use functions declared below to evaluate condition $C$ and measure $F$.

```
float   C(t,float[]);                   /*termination condition          */
float   F(t,float[]);                   /*class 2 output measure         */
```

We categorize termination algorithms based on three criteria; each choice is denoted by a symbol. First, is Interval Termination with Prospective Generation, Exhaustive Termination with Dissociative Generation, or Exhaustive Termination with Retrospective Generation used (I, D, or R)? Second, is each evaluation of $C$ done by 1 processor or more than one (1 or P)? Third, is $C$ evaluated between barriers 1 and 2 or 2 and 3? We do not evaluate $C$ between barriers 3 and 1 because this is where its result, stored in `SV`, is read.

This leads to 12 combinations: IS12, IS23, IP12, IP23, DS12, DS23, DP12, DP23, RS12, RS23, RP12, and PR23.

The code modifications for all combinations is given by Sanjeevan [18]. Class P algorithms are not here studied for the same reason that algorithms EP and IP are not studied in Section 4. In addition, experiments are not done with output measures for the reason given in Section 4. Hence classes D and R algorithms will not differ appreciablely in metric $R$; therefore only class D is examined. Class IS12 is unlikely to outperform class IS23, because some processors are

```
#define B ...                                  /*bounded lag     */
#define M ...                                  /*number of LP's */
#define true  1
#define false 0
typedef boolean int;
typedef Time float;

struct  Event {
  Time  Timestamp;
  ...;
};

/*global, shared data */
boolean SV=false;
Event*  InputQ[L];                             /*input queue of waiting events          */
float   Floor=0.0,                             /*lower bound on current window          */
        Alpha[L],                              /*sim. time when other LP's affect LPi   */
float   S[L],                                  /*Attributes required for C              */
        O[L];                                  /*Attributes required for MG             */

Time    Next(Event* E);                        /*returns 1st event timestamp in E       */
Event*  RemoveHead(Event* E);                  /*removes, returns 1st event in E        */

BoundedLagLP(int i) {
  Event E;                                     /*event in execution or last executed    */
  float t;                                     /*sim. time: last event processed by LP  */
  float n;                                     /*buffers Next(InputQ[i])                */

  while (!SV) {
    compute Alpha[i];                                                          /*S01*/
    synchronize;                                                               /*S02*/
    while ( Next(InputQ[i]) < min(Alpha[i],Floor+B) ) {   /*barrier 1*/       /*S03*/
      E = RemoveHead(InputQ[i]);                                               /*S04*/
      t = E->Timestamp;                                                        /*S05*/
      execute E;                                                               /*S06*/
      optionally insert events into InputQ;   /*may modify S[i],O[i]*/        /*S07*/
  }                                                                            /*S08*/
  synchronize;                                /*barrier 2*/
  if(i==1) Floor = min(Next(InputQ[0]), ..., Next(InputQ[M-1]);                /*S11*/
  synchronize;                                /*barrier 3*/                    /*S12*/
}                                                                              /*S13*/
                                                                               /*S14*/
```

Figure 14: Algorithm for $LP_i$ of underlying bounded lag conservative-synchronous simulation.

guaranteed to be idle between barriers 2 and 3, and hence available for evaluation of $C$; hence we do not study class IS23. This leaves three algorithms, which are stated and studied below: IS23, DS12, and DS23.

Note that rollback is easy to add to a synchronous-conservative algorithm. At the end of the outer loop in Figure 14, a copy is made of all local variables and the input queue. At a later real time during simulation, the saved state can be restored, and the simulation restarted just after barrier 2. The details are given in Sanjeevan [18].

**IS23:**

*Add after S11:*

```
if (i==2) SV=C(t,S);                    /*only LP2 evaluates C    */
```

*Add after S14:*

```
if (i==1) output F(t,O);                /*compute output measures */
```

Evaluating $C$ using algorithm IS23 imposes a negligible overhead for simple global termination conditions because the evaluation is done sequentially by $LP_2$ between barriers 2 and 3 in parallel with the assignment to `Floor`, which is done by $LP_1$.

**DS12:**

*Add to global, shared variables:*

```
struct Node {
  float t;                              /*timestamp                          */
  int   i;                              /*id of LP that writes record        */
  float S;                              /*Stores S[i]                        */
  float O;                              /*Stores O[i]                        */
};                                                                           */
Node* Q[L,2];                           /*Two queues of attribute values per LP */
int    Iteration=0;                     /*Iteration number                   */
void Append(Node* Q,q);                 /*Appends q to end of queue Q        */
```

*Add before S1:*

```
Node* q;
```

*Add after S7:*

```
q = malloc(sizeof(Node));
q.t=t; q.i=i; q.S=S[i]; q.O=O[i];
Append(Q[i,Iteration%2],q);
```

*Add a new LP:*

```
Node*   RemoveHead(Node* Q) /*removes, returns 1st node in Q*/
boolean IsEmpty(Node* Q)    /*returns true when Q=NULL      */
TerminationLP() {
  float copyS[L];
  float copyO[L];
  float t;
  Node* Qsorted;
  Node* q;
  int   m;
  while (!SV) {
    synchronize;
    for (i=0;i<L;i++)
      Insert each node in Q[i,(Iteration+1)%2] into Qsorted in
        ascending order of its t field;
    do
      q = RemoveHead(Qsorted);
      m=q.i; copyS[m]=q.S; copyO[m]=q.O; t=q.t; free(q);
    while (!SV(t,S) && !IsEmpty(Qsorted))
  synchronize;
  Iteration++;                          /*barrier 2*/
  synchronize;
  }                                     /*barrier 3*/
  if (i==1) output F(t,O);
}                                       /*compute output measures */+
```

The algorithm evaluates $C$ one iteration behind the underying simulation. Between barriers 1 and 2, each LP stores its attributes in an element of array Qin timestamped order. Meanwhile, an auxilliary LP sorts the attributes by timestamp order written to Qin the previous iteration by the LP's and evaluates $C$.

## 5.3 Algorithm Performance

We use only the torus benchmark. The colliding pucks cannot be implemented with a conservative-synchronous protocol because there is no lower bound on the next time at which an LP

may execute an event; hence the bound $B$ is zero.

### 5.3.1 Predicted Performance

The simulation time at which T0 holds varies inversely with simulation size because the larger network has a larger number of servers being modeled in parallel. On the other hand, $\tau$ for termination condition T2 is independent of simulation size because all LP's have to execute until simulation time 100,000.

Lubachevsky shows that at least $O(N)$ events are processed for every iteration of the algorithm [13]. The average cost of processing one event is shown to be $O(\log N)$. These facts are of interest as the algorithms are potentially scalable as the simulation and machine size. We make use of these facts to analyze the performance of our algorithms.

**IS23+T0:** For this algorithm, each of the $O(L)$ events executed per iteration takes $O(1)$ time to execute. For T0, the total number of events executed up to termination by all LP's is constant. Thus, for fixed $P$, the total running time of the simulation is $O(1)$ and independent of $L$. Thus we should have a flat curve for the simulation running time as $L$ is varied.

**DS12+T1:** For this algorithm, each of the $O(L)$ events executed per iteration takes $O(1)$ time to execute. For T1, the total number of events executed up to termination by all LP's is constant. The total running time taken up by event execution is $O(1)$. However, for each iteration of the algorithm, there is an $O(L^2)$ cost to sort the events in timestamp order. The number of sorts performed is inversely proportional to $L$ for fixed $P$. So the total running time taken for sorting is $O(L)$. Thus, the the total running time of the simulation is $O(1) + O(L) = O(L)$ and we should get a linear curve for the running time of the simulation as $L$ is varied.

**IS23+T2:** Each of the $O(L)$ events executed per iteration takes $O(1)$ time to execute. For T2, the total number of events executed up to termination by all LP's is $O(L)$, as all LP's simulate up to the same simulation time. Thus, for fixed $P$, the total running time of the simulation is $O(L)$ and we should get a linear curve for the simulation running time as $L$ is varied.

**DS12+T3:** Each of the $O(L)$ events executed per iteration takes $O(1)$ time to execute. For T3, the total number of events executed up to termination by all LP's is $O(L)$. The total running time taken up by event execution is $O(L)$. However, for each iteration of the algorithm, there is an $O(L^2)$ cost to sort the events in timestamp order. The number of sorts performed is independent of $L$ for fixed $P$. So the total running time taken for sorting is $O(L^2)$. Thus, the the total running time of the simulation is $O(L)+O(L^2) = O(L^2)$ and we should get a quadratic curve for the running time of the simulation as $L$ is varied.

### 5.3.2 Measured Performance

Our experiments use a Sequent Symmetry S81 shared-memory multiprocessor with ten 80386 processors running Dynix V3.0.18, AT&T C++ 1.2.1, and the Presto 0.4 thread package version 0.4 [5].

We use $N = 3, 4, 6, 8, 10, 16$ and 20 for the combinations IS23+T0, DS12+T1, IS23+T2, and DS12+T3. We vary $P$ from 1 to 8. Each server is implemented by a thread executing the algorithm with its own copy of all local data. Barrier synchronization is effected using a master-slave mechanism based on examples distributed with Presto 0.4. The value of the bounded lag, $B$, is $1.5x = 150$, so that each node must examine only its nearest neighbors to calculate the earliest time another event could affect its history.

The data is collected while the simulation is the only program running. The difference in $R$ is less than 1% for the extreme values $N$=3 and $N$=20, for $P = 1, 2, 4$, and 8. The simulation is run with both preemptive thread scheduling (with the quantum size varied between 100 and 600 milliseconds) and with non-preemptive scheduling, but this has no effect on $R$, perhaps because no event requires more than 100 milliseconds of real time to execute. The case of $P = 1$ is simply an execution of the parallel simulation on one processor.

The performance of the torus network without a termination algorithm, that is just testing "`Floor > 100000`" in place of "`!SV`" in Figure 14, yields $R$ slightly smaller than the values reported for IS23+T2.

Figure 15 shows the running times for IS23+T0 and DS12+T1. We note that, as predicted, the curve for algorithm IS23+T0 is essentially flat, and the curve for algorithm DS12+T1 is

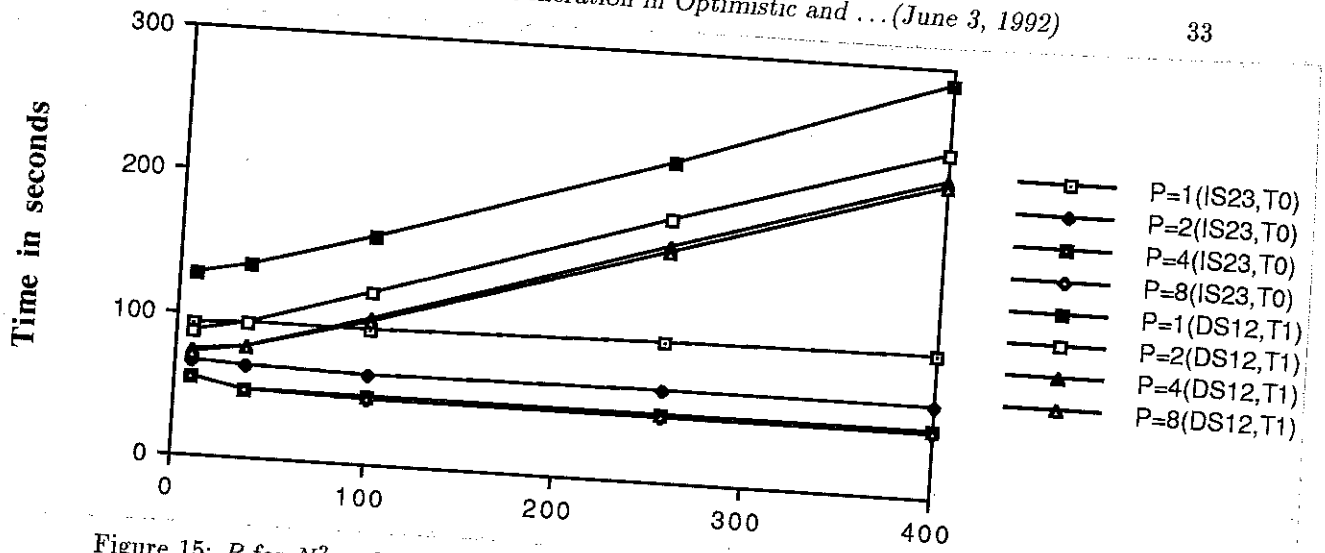



Figure 15: $R$ for $N^2$ node torus network with IS23+T0 and DS12+T1 under bounded lag.

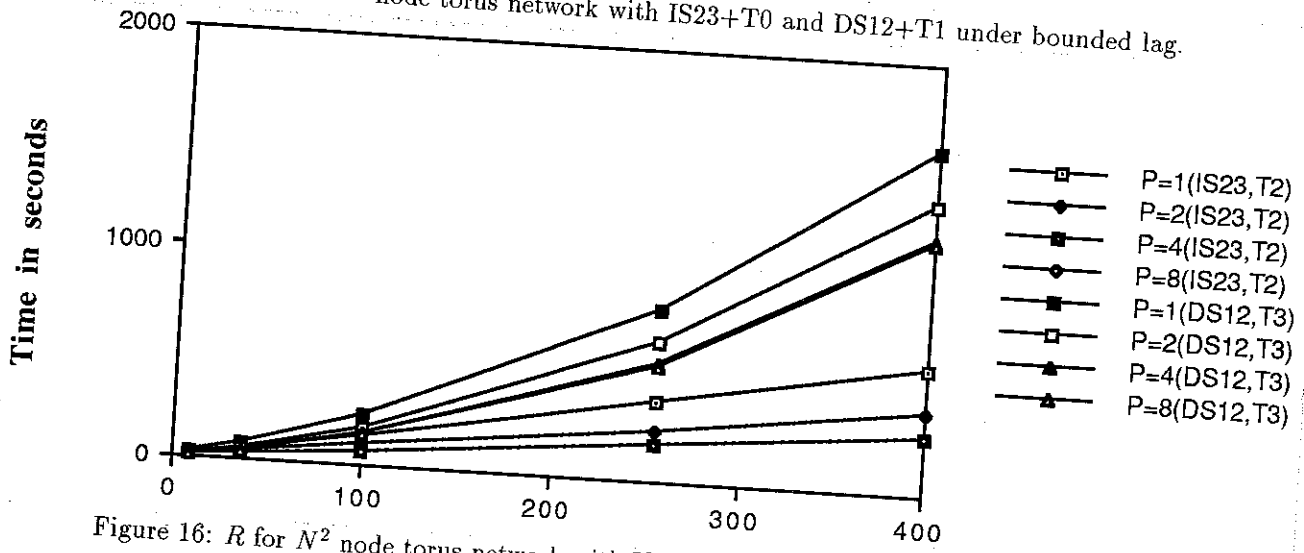



Figure 16: $R$ for $N^2$ node torus network with IS23+T2 and DS12+T1 under bounded lag.

linear. There is an appreciable improvement in performance as $P$ increases from 1 to 2, and from 2 to 4. There is not much improvement if we use 8 processors instead of 4. We conjecture that this is due to the low density of events being processed between barriers 1 and 2, compared to the duration of barrier synchronization.

Figure 16 shows the running times for IS23 + T2 and DS12 + T3. As predicted, the graph for algorithm IS23+T2 is linear and the graph for algorithm DS12+T3 is quadratic. As in the previous graph, we get performance improvements as we increase $P$ from 1 to 2 and from 2 to 4, but not with $P = 8$.

# 6 Comparison to Lin's Algorithms

The termination algorithms (Interval and Exhaustive) and the generation algorithms (Dissociative, Retrospective, and Prospective) proposed here are refinements of algorithms proposed by Abrams and Richardson [1]. Richardson specifies and proves the correctness of the algorithms in a form which applies to any possible space-parallel simulation protocol [16]. Lin gives a detailed description of Exhaustive Termination specifically for time warp [12, Section 3.2]; the algorithm is a refinement of the protocol-independent specification given by Richardson. Lin's algorithm follows the time warp implementation outlined by Abrams and Richardson in that the underying simulation is augmented with an extra LP (e.g., TD) to which the underlying LP's send changes in attribute values required to evaluate $C$. Richardson's proof system (Unity [6]) presumes a fixed number of processes, while Lin's correctness proof does not require this assumption. Lin also gives an alternative implementation of Interval Termination to that proposed here in which the process initiating GVT computation requests attribute values required to evaluate $C$ [12, Section 3.1].

Lin also derives analytic formula for estimating detection delay. Letting $t$ denote the real time when termination is detected, he estimates detection delay $R-t$ by modeling TD as either a FIFO M/G/1 system for heavy traffic or a FIFO G/G/1 system for light traffic, in which customers represent UPDATE messages. Lin's analysis does not consider the overhead added to the underlying simulation by termination *before* termination is detected, which the analysis of Section 1.3 examines. In the measurements of Sections 4 and 5, the overhead added to the underlying simulation is far more significant. In the measurements, detection delay elongates $R$ by a few percent, while the termination algorithm overhead, for unstable conditions, can change $R$ from $O(L)$ to $(L^2)$ (e.g., Figure 16).

# 7 Conclusions

The complexity analysis (Table 1) implies that when the simulation size is sufficiently large and an unstable termination that is a function of attributes private to $O(L)$ LP's is used, where $L$ is the number of LP's, then evaluation of the termination condition will dominate the real

time required to execute the simulation and reduce or even preclude speedup. At this point finding algorithms to evaluate the termination condition (e.g., using parallel evaluation of $C$, use of multiple TD's) become more important than parallelizing the underlying simulation. For example, a change of a single word in a termination condition (T2 versus T3 for the torus benchmark) can change the time complexity of simulation from $O(L)$ to $O(L^2)$ (Figure 16). Other unstable condition graphs (Figures 11 and 16) are inconclusive this point, because the simulation sizes used are insufficient to identify whether the curves are linear or are the initial portions of exponential curves. Storage requirements of the simulations precluded measurement of larger simulation sizes to confirm the graph shapes. In all cases termination increases the simulation running time, and for unstable conditions by a substantial amount.

Output measure generation is a less severe problem. It can be done with zero additional space using rollback, or by restarting the simulation in protocols without rollback, by roughly doubling the real time required for simulation. Alternately, a conservative-synchronous protocol can do measure generation in additional space proportional to the window size without increasing the real time required for simulation. However, one might limit the window size in a conservative-synchronous protocol, possibly increasing the real time required for simulation, to limit the space required for measure generation.

Conservative-synchronous and optimistic simulations appear to be equally amenable to the addition of termination and measure generation algorithms. Conservative-synchronous protocols require somewhat more programming effort for unstable conditions, because one must add a rollback mechanism. However rollback is simpler in conservative-synchronous protocols than in optimistic protocols. Conservative-asynchronous protocols require at worst unbounded space, and are recommended only when a stable condition and class 2 measures are used.

From the complexity analysis, the termination cost dominates the real time required to run a simulation when the real time required for each evaluation of $C$ exceeds the real time required to execute an event. Therefore for an unstable condition there should exists a critical value of simulation size, such that when the size is exceeded, the real time required for simulation should dramatically increase. The graphs in the paper do not exhibit this phenomena, but this is probably due to the fact that the simulation sizes we use in this paper are relatively small

– no more than 128 pucks or 400 queueing network servers. In contrast, in a simulation with thousands to millions of LP's, the cost of termination may become a fundamental limitation on speedup.

## Acknowledgments

The authors wish to express their thanks to Peter Reiher for arranging use of the Jet Propulsion Lab's TWOS and colliding pucks simulation for the measurements in Section 4. The bounded lag measurements were made on the Sequent Symmetry at the Argonne National Laboratory.

## References

[1] M. Abrams and D. S. Richardson, "Implementing a Global Termination Condition and Collecting Output Measures in Parallel Simulation," in *Proceedings of the 1991 Workshop on Parallel and Distributed Simulation*, Anaheim, CA, Jan., 86-91.

[2] M. Abrams, *Terminating Parallel Simulations*, TR 92-1, Computer Science Department, Virginia Tech, Blacksburg, Virginia, Jan., 1992.

[3] R. Ayani, "Parallel Simulation using Conservative Time Windows," to appear in *Winter Simulation Conference*, Dec. 1992.

[4] O. Balci, "The Implementation of Four Conceptual Frameworks for Simulation Modeling in High-Level Languages." in *Proc. of the 1988 Winter Simulation Conference*, San Diego, CA, Dec., 287-295.

[5] B. N. Bershad, E. D. Lazowska, and H. M. Levy, *PRESTO: A System for Object-Oriented Parallel Programming*, Technical Report 87-09-01, Department of Computer Science, University of Washington, Seattle, Jan. 1988.

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

[7] K. M. Chandy, and R. Sherman, "Space-Time and Simulation," in *Distributed Simulation*, San Diego, Calif, Jan. 1989, 53-57.

[8] R. M. Fujimoto, "Parallel Discrete Event Simulation," *Comm. ACM 33*, (10), Oct. 1990, 30-53.

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

[10] D. R. Jefferson, P. Hontalas,*et al*, "Performance of the Colliding Pucks simulation on the time warp operating system (Part 1)," in *Proc. of Distributed Simulation 1989*, Tampa, FA, pp. 3-7.

[11] T. V. Lakshman and V. K. Wei, "On Efficiently Computing Functions of Distributed Information," submitted for publication.

[12] Y. Lin, *On Terminating a Distributed Discrete Event Simulation*, submitted for publication.

[13] B. Lubachevsky, "Efficient distributed event-driven simulations of multiple loop networks," *Comm. ACM 32*, Jan. 1989, 111-123.

[14] F. Mattern, "Algorithms for Distributed Termination Detection," *Distributed Computing 2*, 1987, 161-175.

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

[16] D. S. Richardson, *Terminating Parallel Discrete Event Simulations.* M.S. thesis, TR 91-9, Computer Science Department, Virginia Tech, Blacksburg, Virginia, May, 1991.

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

[18] V. Sanjeevan, *The Cost of Terminating Conservative-Synchronous and Optimistic Simulations.* M.S. thesis, Computer Science Department, Virginia Tech, Blacksburg, Virginia, June, 1992.

[19] Sokol, L.M., Briscoe, D. P. and Wieland, A.P. 1988. "MTW: A Strategy for Scheduling Discrete Simulation Events for Concurrent Execution," in *Proc. of the SCS Multiconference on Distributed Simulation*, 19(3), July 1988.