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

# PARALLEL SIMULATION USING CONSERVATIVE TIME WINDOWS

Rassul Ayani
Hassan Rajaei

Department of Telecommunication and Computer Systems
Royal Institute of Technology,
S-100 44, Stockholm, Sweden

## ABSTRACT

This paper, presents a Conservative Time Window(CTW) algorithm, for parallel simulation of discrete event systems. The physical system to be simulated is partitioned into n disjoint sub-systems, each of which is represented by an object. The CTW-algorithm identifies a time window for each object, such that events occurring in each window are independent of events in other windows and thus they can be processed concurrently. The CTW-algorithm was implemented on a shared memory multiprocessor, a Sequent Symmetry S81 with 16 processors. We measured performance of the CTW algorithm on two types of network topologies, feed-forward networks and networks with feedback loops. We used three metrics to measure performance: Speedup, Average Number of Independent Windows detected by the algorithm, and Average Number of Events occurring in each Window. The obtained results suggest that the performance of the CTW algorithm is good for certain classes of applications.

## 1. INTRODUCTION

Two main paradigms, so called optimistic and conservative approach, have been proposed for distributed discrete-event simulation. The optimistic paradigm (Jefferson 1985) requires both time and space for saving state variables and performing roll back (Fujimoto 1989, Righter and Walrand 1989), while the conservative paradigm (Chandy and Misra 1979, Misra 1986, Peacock, Wong, and Manning 1979) is vulnerable to deadlock and memory overflow in general (Misra 1986, Wagner, Lazowska, and Bershad 1989).

Lubachevsky (1988) suggests a conservative approach to parallel simulation, where an incoming (outgoing) spherical region is defined for each node. This region is combined with a bounded lag restriction to determine a set of safe events which can be processed concurrently.

Nicol (1991) has proposed another conservative protocol which is in many aspects similar to the one proposed by Lubachevsky. In this protocol, each LP may advance its time up to a *global ceiling*. We discuss in more detail the relationship of the work of Lubachevsky and Nicol to our scheme in Subsection 5.3.

Another approach to parallel simulation has been introduced by Sokol, Bristo, and Wieland (1988), using Moving Time Windows (MTW). In this approach, a global time window, which is dynamically adjusted, is assigned to the nodes. Events within the time windows are *assumed* to be parallel. An anomaly occurs when a node schedules an event earlier than the recipient node's latest executed event. The essential point of this approach is that the authors relax some of the precedence constraints of the traditional sequential simulation. Consequently, result of MTW scheme is not necessarily identical to the results of the sequential one.

The principal contribution of our paper is to present a parallel simulation scheme which employs Conservative Time Windows (CTW) and to evaluate its performance. The system to be simulated is partitioned into n disjoint sub-systems, each of which is represented by an object. The scheme identifies a time window for each object such that events within these windows are *independent* and can be processed *concurrently*. In our scheme presented in this paper, the size of each window is calculated in each iteration of the algorithm using feature of the system being simulated. Thus, different windows may have *different sizes* if the nodes are advancing heterogeneously As opposed to other similar methods, e.g. Lubachevsky (1989), Nicol (1991), the windows are not bounded from above by a global *ceiling*. We will show that this feature of the CTW-algorithm exploits more parallelism compared to the case where a global ceiling must be kept by all nodes. We conducted extensive experiments to measure performance of the CTW-algorithm. Our results show that the CTW-algorithm gives a significant speedup over a good sequential simulator in many cases.

The rest of this paper is organized as follows: First, some definitions are given in section 2, and then the CTW-algorithm is presented in sections 3 and 4. Performance measurement results are presented in section 5. Finally, some conclusions are given in section 6.

## 2. PRELIMINARIES

The system to be simulated is partitioned into $n$ disjoint sub-systems, each of which is represented by an object. Thus, the simulation system consists of $n$ objects, denoted by $O_1$, $O_2$, ..., $O_n$ which communicate with each other at discrete times. Each object $O_i$ has its own local time $LT_i$ and maintains its own event list $EL_i$.

**Definition 1**: The **directional distance** of an object $O_i$ to another object $O_j$ of the system, denoted by $d_{ij}$, is defined as the **lower bound** on the simulated time delay between the occurrence of any possible event in $O_i$ and its possible effect on $O_j$.

**Definition 2**: Event $x$ may **affect** event $y$, denoted by $x \rightarrow y$, if $x$ schedules an event $z$ for $O_j$ which occurs before $y$, i.e $t(z) < t(y)$. Otherwise $x$ **cannot** affect $y$, denoted by $x \not\rightarrow y$.

Considering the relation between $x$ and $y$, where $x$ is in $O_i$ and $y$ is in $O_j$, the following cases may arise:

i) $t(x) + d_{ij} < t(y)$.

Event $x$ *may* schedule an event $z$ for $O_j$ with $t(z)=t(x)+d_{ij} < t(y)$. Hence, according to definition 2, $x$ may *affect* $y$. Thus,

$$t(x) + d_{ij} < t(y) \Rightarrow x \rightarrow y \qquad (1)$$

ii) $t(x) + dij \geq t(y)$.

Suppose that execution of event $x$ schedules an event $z$ for $O_j$. Since $d_{ij}$ is the lower bound for the time delay (definition 1), it follows that:

$$t(z) \geq t(x) + d_{ij} \geq t(y)$$

Consequently, processing $x$ *cannot cause any effect* on $y$. Thus,

$$t(x) + d_{ij} \geq t(y) \Rightarrow x \not\rightarrow y \qquad (2)$$

**Definition 3**: Two events $x$ and $y$ are said to be **independent** if they do not affect each other, denoted by

$$x \not\rightarrow y \text{ and } y \not\rightarrow x \Rightarrow x \not\leftrightarrow y \qquad (3)$$

Thus,

$$t(x) + d_{ij} \geq t(y) \text{ and } t(y) + d_{ji} \geq t(x) \Rightarrow x \not\leftrightarrow y \qquad (4)$$

Denote the first event in the event list of an object $O_i$ by $e_i$, i.e.,

$$t(e_i) = \min_{x \in EL_i} \{ t(x) \} \qquad (5)$$

Thus,

$$\text{If } t(e_i) + d_{ij} \geq t(e_j) \text{ and } t(e_j) + d_{ji} \geq t(e_i), \text{ then } e_i \not\leftrightarrow e_j \qquad (6)$$

These relations can be used to detect concurrent events in a system, as is discussed later.

## 3. CONSERVATIVE TIME WINDOWS (CTW)

A conservative parallel simulation scheme, called three phase algorithm (TPA), has been proposed by Ayani (1991). The TPA, as sketched in Figure 1, identifies at most one event per object in each of its iterations (Ayani 1991). Since the CTW-algorithm is an extension of the TPA, we begin by discussing the TPA.

```
repeat
   1) Nomination Phase:
      Mark the first event of each object as a
      candidate for concurrent evaluation.
   Barrier
   2) Concurrency Control Phase:
      Identify independent events among the
      marked ones using (6)
   Barrier
   3) Evaluation Phase:
      Process the independent events.
   Barrier
Until (End-of-Simulation)
```

**Figure 1**: The Three Phase algorithm based on distances between objects

**Definition 4**: A time window is simply a time interval [L, U]. A time window for $O_i$ is denoted by $W_i=[L_i, U_i]$.

In this paper, we present an algorithm which produces $n$ time windows (one per object) in each of its iteration.

**Definition 5**: Two time windows $W_i$ and $W_j$ belonging to two different objects $O_i$ and $O_j$ are said to be independent if

$$x \not\leftrightarrow y \; \forall \; (x, y) : t(x) \in W_i \text{ and } t(y) \in W_j$$

**Definition 6**: Similarly, $k$ windows $W_1$, $W_2$, ..., $W_k$ are said to be independent if they are pairwise independent.

**Lemma 1**: Assume that $W_i = [t(e_i), U_i]$ and $W_j = [t(e_j), U_j]$ are two time windows for $O_i$ and $O_j$ respectively (Figure 2). If

i) $t(e_i) + d_{ij} \geq U_j$ and

ii) $t(e_j) + d_{ji} \geq U_i$

then $W_i$ and $W_j$ are independent.

**Proof**: For any pair $(x, y)$: $x \in EL_i$, $t(x) \in W_i$ and $y \in EL_j$, $t(y) \in W_j$, $i \neq j$, $i, j = 1,...,n$,
we have: $U_i \geq t(x) \geq t(e_i)$ and $U_j \geq t(y) \geq t(e_j)$.
Since $d_{ij} \geq 0 \Rightarrow$

$$t(x) + d_{ij} \geq t(e_i) + d_{ij} \geq U_j \geq t(y)$$

and similarly,

$$t(y) + d_{ji} \geq t(e_j) + d_{ji} \geq U_i \geq t(x)$$

and thus according to (4) these two events are independent. ♦

Lemma 1 states that if $W_i$ and $W_j$ are two time windows for $O_i$ and $O_j$ respectively, such that (1) processing $e_i$ does not affect $O_j$ (directly or indirectly) before simulation time $U_j$ and, (2) processing $e_j$ does not affect $O_i$ before $U_i$ (directly or indirectly, as illustrated in Figure 2), then $W_i$ and $W_j$ are two independent windows.

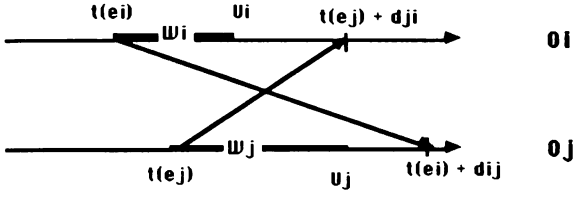


**Figure 2**: Two Independent Time Windows

**Lemma 2**: Suppose that $W_j = [t(e_j), U_j]$ is a time window for $O_j$, $j = 1, 2, ..., k \leq n$ and

$$U_j = \min_{\substack{1 \leq i \leq k \\ i \neq j}} \{t(e_i) + d_{ij}\} \qquad (7)$$

then $W_1, W_2, ..., W_k$ are independent.

**Proof**: For any pair $(x, y)$: $x \in EL_i$, $t(x) \in W_i$ and $y \in EL_j$, $t(y) \in W_j$, $i \neq j$, $i, j = 1,...,k$,
we have: $U_i \geq t(x) \geq t(e_i)$ and $U_j \geq t(y) \geq t(e_j)$.
Since $t(x) \geq t(e_i) \Rightarrow t(x) + d_{ij} \geq t(e_i) + d_{ij}$
Using (7) $\Rightarrow t(x) + d_{ij} \geq t(e_i) + d_{ij} \geq U_j$
Similarly

$$t(y) + d_{ji} \geq t(e_j) + d_{ji} \geq U_i$$

which (according to Lemma 1) implies that $x$ and $y$ are independent.
Since $x$ and $y$ are arbitrary in $W_i$ and $W_j$ respectively, it follows that $W_i$ and $W_j$ are independent. ♦

This Lemma states that if $W_1, W_2, \ldots, W_k$ are $k$ time windows, where $U_j$ is the lowest timestamp that object $O_j$ may receive, then these windows are independent.
Based on Lemma 2, a time window $W_i$ can be calculated for each object $O_i$ ($W_i$ may be empty), such that any event $x \in EL_i$ with $t(x) \in W_i$ is independent of any other event $y \in EL_j$ with $t(y) \in W_j$, for $i, j = 1, 2, ..., n$; $i \neq j$.

To calculate these windows, each object $O_i$ first nominates a window $W_i = [t(e_i), MAX\_SIM\_TIME]$, where MAX_SIM_TIME is the simulation time up to which the system will be simulated.

Second, the length of the windows is adjusted by checking the possible effect of processing $e_i \in EL_i$ on other objects. This can be done by computing $U_j$ according to (7) and then comparing it with $t(e_i)$. Thus, if $U_j < t(e_j)$ for an object $O_j$, then an empty window $W_j = []$ is assigned to $O_j$, otherwise the window size is adjusted, by setting $W_j = [t(e_j), U_j]$.

Finally, events in different windows are processed concurrently and new events (if any) are inserted in the event list of the objects. A formal description of this algorithm is given in Figure 3.

Figure 4 illustrates some iterations of the CTW-algorithm for a system containing 3 objects, where $W_{ij}$ denotes a time window belonging to $O_i$ at iteration number $j$. As can be seen, the size of the windows may change radically and the local times of the objects move forward asynchronously.

The CTW-algorithm has the following properties:

a) Events within a window are processed sequentially, but, events within different Windows are independent and can be processed concurrently.
b) The size of each window is dynamically determined. Different windows, even those belonging to the same iteration of the algorithm, may have different sizes (as shown in Figure 4).
c) Each of the *three phases* of the algorithm may be executed by several processors in parallel. However, synchronization is required between any two consecutive phases.

```
Initialization stage : initialize the event lists and
other system variables
repeat
   1)   Nomination Phase:
      for i = 1 to n do
         begin
            Wi = [t(ei), MAX_SIM_TIME];
         end
   Barrier

   2)   Adjustment Phase:
      Identify independent windows as follows:
      for j = 1 to n do
         begin
            Uj = min{min{t(ei) + dij}, MAX_SIM_TIME}
                 1 ≤ i ≤ n
                 i ≠ j
            If Uj < t(ej) then Wj = []; else Wj = [t(ej),Uj];
         end
   Barrier

   3)   Evaluation Phase:
      Process the events within these windows and insert
      new events (if any).
   Barrier
Until (End-of-Simulation)
```

**Figure 3**: The CTW-algorithm

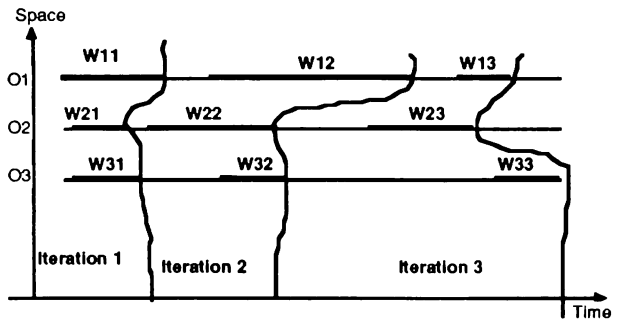


**Figure 4**: Results of performing 3 iterations of of the CTW-algorithm

## 4. IMPROVING THE CTW-ALGORITHM

Up to now it was assumed that each object, $O_i$, maintains its directional distances to all other objects of the system. The CTW-algorithm based on this assumption suffers from the following drawbacks:

i) It restricts scalability of the model, since it is difficult to calculate all the directional distances in a large network, possibly with several hundreds of nodes.

ii) The adjustment phase of the algorithm (Figure 3) requires an order of $n^2$ comparisons to adjust the upper bounds of the windows.

Therefore, it is desirable to replace this rather impractical constraint by a more realistic one.

### 4.1 The Immediate Successor Constraint

Assume that each object maintains its directional distances to its *immediate successors only*. Suppose that $O_1$, $O_2$, and $O_3$ are three objects such that $O_3$ is a second order successor of $O_1$, as illustrated in Figure 5.

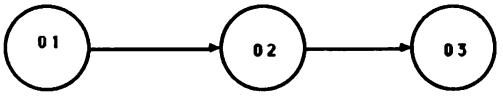


**Figure 5**: A tandem network

Further, suppose that $e_1$, $e_2$, and $e_3$ are events occurring in $O_1$, $O_2$, and $O_3$ respectively. Consider the effect that processing $e_1$ may have on $e_3$:

**case1**: $t(e_1) + d_{12} \geq t(e_2)$. This means $e_1$ cannot affect $O_2$.

**case2**: $t(e_1) + d_{12} < t(e_2)$. This implies that processing $e_1$ *may* schedule an event, with time $t(e_1) + d_{12}$, in $EL_2$, which in turn *may* schedule an event in $EL_3$ prior to $e_3$. However this may happen only if

$$t(e_1) + d_{12} + d_{23} < t(e_3).$$

Thus in the latter case, processing $e_1$ raises the *possibility* that an event is inserted in $EL_2$ prior to $e_2$, the effect of which on $e_3$ must be checked.

Generalization of this observation leads to a modified CTW-algorithm as illustrated in Figures 6 and 7. The implementation details of this algorithm, such as how to access shared data structures, are not given in this paper.

## 5. PERFORMANCE EVALUATION

The purpose of this section is to investigate behavior of the CTW algorithm under various conditions. Some researchers, e. g.,Fujimoto (1989), Nicol (1988), Reed, Malony, and McCredie (1988), Wagner, Lazowska, and Bershad (1989), have shown that network topology, message population, timestamp increment, routing policy, and network size are among the most important factors affecting performance of parallel simulation schemes. We followed a similar approach and measured the result of changing these parameters on the performance of the CTW-algorithm. We conducted extensive experimental measurements on two main types of topologies: feed-forward networks and networks with feedback loops. The experiments were performed on a Sequent Symmetry S81 shared memory computer with 16 processors. The simulation tasks were specified and executed in the SIMA parallel simulation environment (Rajaei 1992).

```
repeat
   1) Nomination Phase:
      for i = 1 to n do
        begin
          min_time_i = t(e_i); U_i = MAX_SIM_TIME;
          W_i = [t(e_i), U_i]; cand[i] = true;
          /* cand[i] = true if W_i is non-empty and false
             otherwise. */
        end
   Barrier
   2) Adjustment Phase
      for i = 1 to n do
        begin
          if cand[i] = true then
            begin
            /*Check the effect of processing e_i on O_i's
               immediate successors by calling: */
               control_event(i, t(e_i))
            end
        end
   Barrier
   3) Evaluation Phase:
      /* Events within different windows are independent
         and can be processed concurrently. */
      Process the parallel events and insert new events (if
      any).
   Barrier
Until (End-of-Simulation)
```

**Figure 6:** A generalized version of the CTW-algorithm

```
control_event(i, t) ::
/* i = obj_number, t = time of an event in  O_i*/
begin
   for all immediate successors, j of O_i do
     begin
      if t + d_ij < U_j then U_j = t + d_ij;
      if t + d_ij < min_time[j] then
         /* an event with time t may be scheduled for
         O_j and thus replace W_j by an empty window */
         begin
            lock[j];
            /* changes are done atomically, so check once
               more if another object made the changes*/
            if t + d_ij < min_time[j] then
               begin
                  cand[i] = false; min_time[j] = t + d_ij ;
                  unlock[j];
                  control_event(j, min_time[j]);
               end;
            else
               unlock[j];
         end
     end
end
```

**Figure 7:** Details of the adjustment phase

We used the following metrics to evaluate the performance of the algorithm:

♦ **Speedup** $S_p$: $S_p = T_s / T_p$ , where $T_p$ is the execution time of the modified CTW-algorithm using $p$ processors, and $T_s$ is the execution time of a sequential simulator. In order to obtain realistic performance figures, a conventional sequential simulator was also developed and used as the basis for all performance comparisons. The event list in the sequential simulator is represented by a *binary heap*. The *binary heap* gives O(log$n$) searches for each insertion (or deletion), compared to O($n$) searches required by a linked list data structure.

♦ **Average Number of Independent Windows (ANIW)**: Those windows that contain at lease one event and are eligible to be processed in the evaluation phase of the algorithm (Figure 3) are counted as independent. Thus, ANIW can be used as a measure for the amount of parallelism detected by the CTW algorithm.

♦ **Average Number of Events per Window (ANEW)**: At the beginning of the evaluation phase each **Independent** window contains one or more events. The number of events occurring in different windows may vary radically. The average number of events occurring in an independent window is denoted ANEW.

### 5.1 Feed-forward Networks

As benchmark for this type of topologies, Multistage Interconnection Network, MIN, was used (Figure 8). The size of the network was varied between 4 and 9 stages (16 to 512 inputs). Each switch of the network was represented by an object. Messages were generated by source nodes attached to inputs, assuming a Poisson distribution. The arrival rate was assumed to be 1/3 for each input link (2/3 for each input switch) in most of the experiments. However we also performed some limited experiments with arrival rate 1/2. The simulation time for all experiments was assumed to be 100,000 units. The service time of a message $j$ at a node $i$, $ST_{ij}$, was defined as:

$$ST_{ij} = c_i + v_j$$

where

$c_i$ is the constant part of the service time for node $i$; it can be considered as the minimum time a node needs to process a message, and

$v_j$ is the variable part of the service time depending on the message $j$.

#### 5.1.1 Symmetric Workload

In this set of experiments, we assumed that the message destination is uniformly distributed among the N outputs of the MIN; and whenever a message reaches its destination it is discarded. Hence, the total number of messages in the network may change. The values of $c_i$ and $v_j$ are calculated in the following way: $c_i$ = C + expntl(A); $v_j$ = expntl(B), where A, B, and C are some constants. In this way, C is a common constant, but expnl(A) generates an individualized value for each of the objects. Thus, in general, $c_i \neq c_k$, for $i \neq k$. Similarly, $v_j$ values are different for different messages. The variable $v_j$ is exponentially distributed with mean B.

We studied, among others, the impact of various values of A, B, and C on performance of the CTW-algorithm.

Figure 9 illustrate the relationship between the size of the network and its performance for some values of A, B and C. As can be seen, the size of the network has a substantial impact on the speedup.

Figure 10 illustrates the relation between some of the simulation parameters and the ANIW and Figure 11 for the ANEW. As shown by the figures, the variation of B and C does not have much impact on the ANIW and ANEW. However, the size of the network has a significant effect on the ANIW.

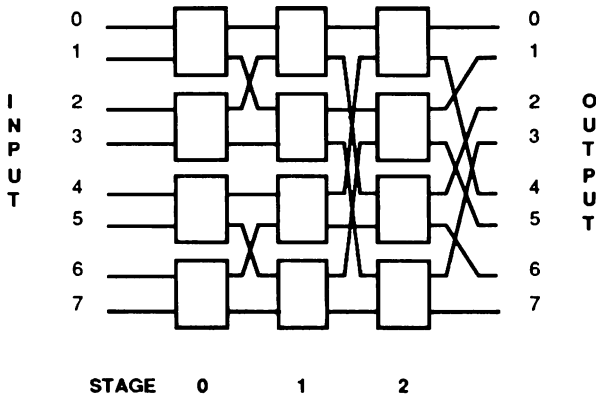


**Figure 8**: A Multistage Interconnection Network with 8 inputs

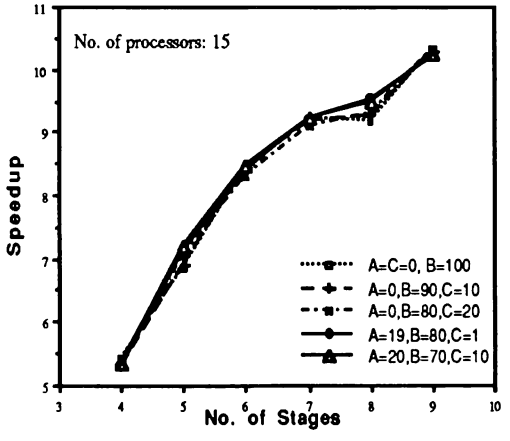


**Figure 9**: Impact of network size on speedup

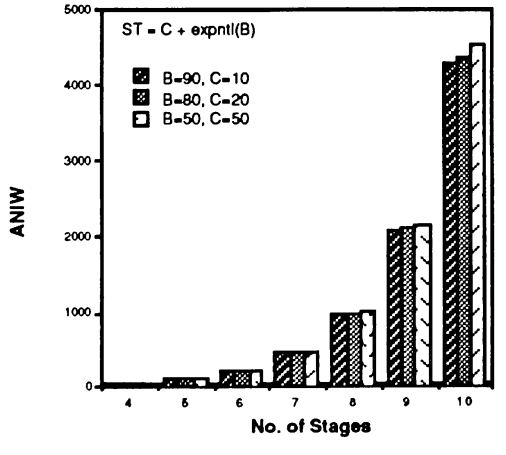


**Figure 10**: Average Number of Independent Windows detected by the CTW-algorithm in each iteration

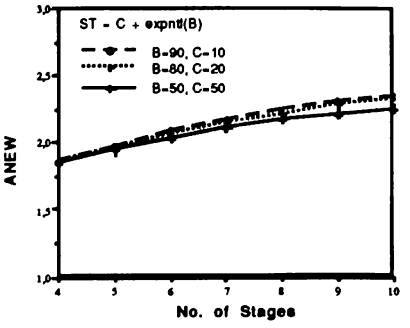


**Figure 11**: Average Number Events occurring in each Windows.

### 5.1.2 Asymmetric Workload

To measure the performance under asymmetric workload, we assume that $P$-percent of the messages are sent to a *hot-spot* node. In our experiment, we varied the value of $P$ between 0 and 100. For the sake of simplicity, this experiment was carried out with a constant service time of 10 assigned to all nodes. Figure 12 illustrates the impact of $P$ on the speedup. The speedup is stable for low values of $P$, but it drops considerably when the number of hot-spot messages reaches certain level. We may call this value of P as *turning-point*, which is equal to 20 in our experiment (Figure 12).

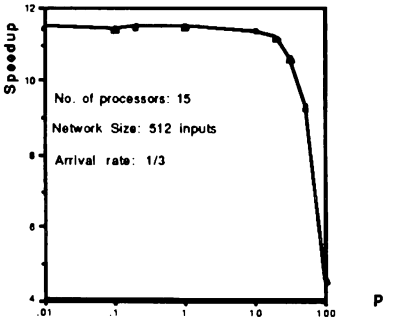


**Figure 12**: Obtained speedup when $P$% of messages are sent to a hot-spot node

## 5.2 Networks with Feedback Loops

A torus network (Figure 13) was chosen as benchmark for this type of topologies. The size of the network was varied from 4x4 to 20x20. Messages were initially generated for each node and sent out to one of the output links of the node. Hence, the message population was kept constant during the simulation. We experimented with cases where 1, 4, 10, 50, and 100 initial messages were generated for each node.

The service time for a node $i$ was defined as:

$$ST_i = c_i + v_i$$

where

$c_i$ is the constant part of the service time, and

$v_i$ is the variable part of the service time.

We assumed that $c_i$ = C + expntl(A) and $v_i$ = expntl(B). The slight changes in the definition of the service time (compared with 5.1) was aimed at making comparison of our results with those reported in the literature easier.

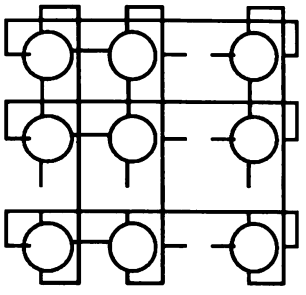

**Figure 13**: A torus network with dimension $d$ (number of nodes: $d$x$d$)

### 5.2.1 Symmetric Workload

In this set of experiments, we assumed that messages are routed uniformly to one of the four neighbours of a node. We studied the impact of network size, message population and the computation time required for a message on speedup.

Figure 14 illustrates the impact of network size and number of processors on speedup. The speedup increases almost linearly for large torus networks, but much slower for small ones (e.g., see the values for the 4x4 torus).

We introduced an artificial processing time, $t$, to each message. The value of $t$ corresponds to the time required for processing an event in a real simulation. Thus, $t$ can be seen as *event granularity*. Figure 15 illustrates the impact of $t$ on speedup, where $d_{ij}$ =10, for all $i$ and $j$.

We also investigated the impact of network size and message population on number of independent time windows detected by the CTW algorithm. Figure 16 illustrates ANIW (which corresponds to the degree of parallelism detected by the CTW-algorithm). As can be seen, ANIW depends heavily on network size.

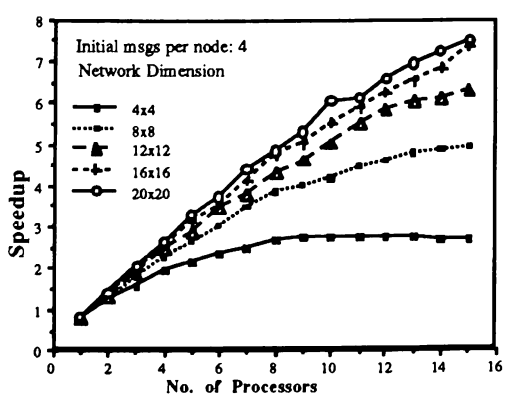


**Figure 14**: The impact of network size on speedup, where $d_{ij}$ = 10 for all $i$ and $j$, i.e. A = B = 0, and C = 10

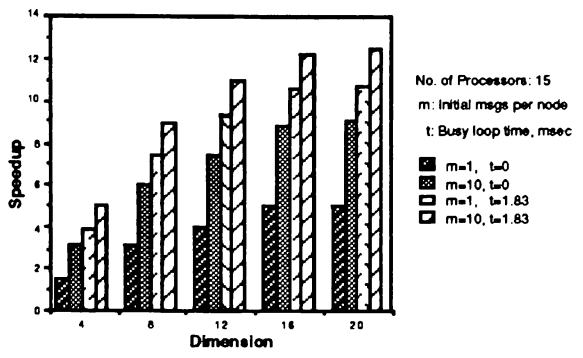


**Figure 15**: The impact of computation time needed by a message on speedup, where $d_{ij}$ = 0 for all $i$ and $j$, i.e. A = B = 0, and C = 10

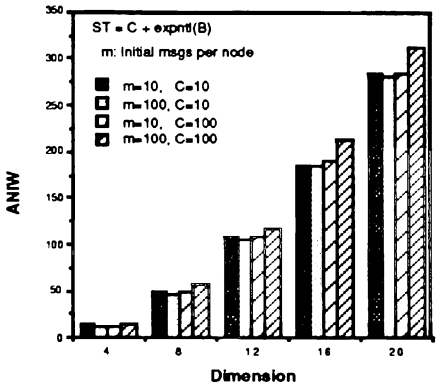


**Figure 16**: Average Number of Independent Windows detected in each iteration of the CTW-algorithm for different network sizes and message populations

### 5.2.2 Asymmetric Workload

We introduced some lazy nodes in the torus network to study the asymmetric case. We assumed that a lazy node is 10 times slower than a normal node. The number of lazy nodes has been varied between 0 (no lazy node) and 16, where 16 is the dimension of the torus network used in these experiments. Figure 17 shows that the speedup decreases when the number of lazy nodes are increased. This figure indicates that the drop in the speedup depends heavily on the message population.

## 5.3 Comparison with Related Works

Boris Lubachevsky has proposed a bounded lag algorithm for parallel simulation (Lubachevsky 1988 and Lubachevsky 1989). Our approach is similar to his in many aspects. He also uses the distance between objects (referred to as minimum propagation delay) and several phases separated by barriers. However, he calculates an incoming (outgoing) spherical region for each node and uses these regions, in combination with a bounded lag (BL) restriction, to determine a set of safe events. Lubachevsky also calculates a *global floor*, which is the

minimum of all event times, in each cycle of his algorithm, and broadcasts this floor to all objects. His approach uses a *global* window for all objects (ceiling of the window = floor + BL). The size of BL has a significant impact on performance (e.g. see Figure 18),

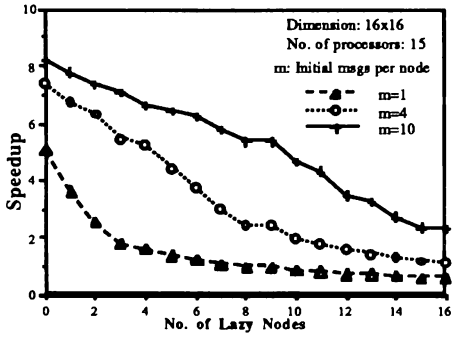


**Figure 17**: The impact of having k lazy nodes on the speedup

but it must be specified by the user; and no method has been proposed to find an optimal value. Moreover, calculation and broadcasting of the floor requires an additional overhead which depends on the number of nodes. In our approach: (a) All windows are *local*, (b) The width of each window is *dynamically* determined in each iteration of the algorithm using a recursive function (Figures 6 and 7) which is based on Lemma 2. (c) Different windows, even those belonging to the same iteration of the algorithm, may have *different sizes* (as shown in Figure 4) and the windows are not cut from above by a global ceiling. Our experiments indicate that the latter feature of the algorithm exploits more parallelism as opposed to the case where a global ceiling must be kept (see Figure 18).

Nicol (1991) has proposed a conservative protocol where each LP (corresponding to an object in our notation) may advance its time up to a *global ceiling*. However, calculation of the ceiling is different from the one proposed by Lubachevsky.

Although an accurate comparison of our performance results with the related works requires having the same testbed and the same parameters, we try to correlate our results with some of those reported in the literatures, in particular with (Lubachevsky 1989, Nicol 1991).

Lubachevsky reports an efficiency of about 50% corresponding to a speedup of 4.5 for simulating a 20x20 torus network using 9 Processors (Figure 11b in Lubachevsky (1989)). We have obtained speedup of 5.5 for a similar case.

To evaluate the impact of BL on performance, we introduced a BL restriction in the CTW-algorithm and conducted several experiments. The experiments were performed on a 16x16 torus network with exponentially distributed timestamps. The results of these experiments (Figure 18) indicate that the size of BL has a significant effect on performance. In these experiments, we did not use any global floor, but it would be interesting to study the cost of calculating and broadcasting it. It should be mentioned that Lubachevsky uses the BL parameter to reduce the cost of identifying safe events, but it was not considered here. Our conclusion is that BL and floor can be used in certain applications where the cost related to using them is less than the gained benefits.

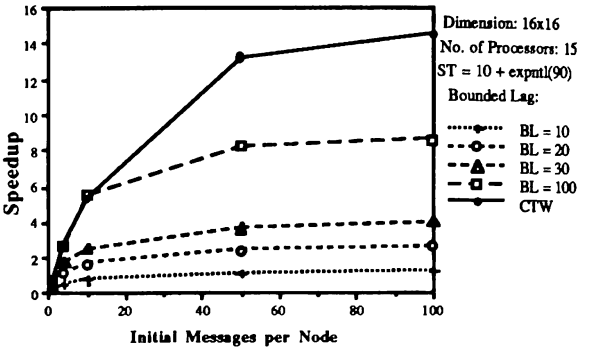


**Figure 18**: The speedup obtained by imposing a Bounded Lag (BL) constraint on the windows is compared to the CTW-algorithm. A 16x16 torus network with uniform routing is assumed.

Nicol has calculated, among others, processor utilization of his protocol for self-initiating networks and has determined where such a conservative protocol can achieve better performance than Time Warp (Nicol 1991). However, the *global ceiling* used in his protocol is an important factor limiting performance of the scheme. For instance, he reports 0.348 processor utilization for K = 32 messages per cycle (Nicol 1991, Table I). Whereas our experimental result shows processor utilization of 0.67 when the CTW-algorithm with a similar set of parameters is used. In our experiment, the overhead assumed by Nicol (1991) was replaced by the actual one.

## 6. CONCLUSIONS

The CTW-algorithm described in this paper is a conservative approach to parallel simulation. In this approach, the system to be simulated is partitioned into $n$ disjoint sub-systems and each sub-system is represented by an object. The algorithm produces a Time Window, which may be empty, for each object. The width of the windows is calculated in each iteration of the algorithm and may be different for different objects (see Figure 4). The number of non-empty windows produced in each iteration of the algorithm and the size of each one depends on features of the system being simulated and the used parameters, e.g. message population, network topology, and network size. A significant feature of the CTW-algorithm is that it, in contrast to other window based methods reported in the literature, e.g. Lubachevsky (1989), Nicol (1991), Sokol, Briscoe, and Wieland (1988), produces *local* bounds for the time windows. Our experiments indicate that this feature improves performance of the algorithm.

We conducted extensive experimental measurements on two main types of topologies, feed-forward networks and networks with feedback loops. The purpose of these experiments was to gain better understanding of the time window protocols in general, and to evaluate performance of the CTW-algorithm in particular. We used mainly three metrics to measure the performance: Speedup, ANIW, and ANEW. The obtained results suggest that:

i) The ANIW depends heavily on message population and network size (see Figures 9 and 15).

ii) The speedup does not depend much on $d_{ij}$ for feed-forward topologies.

iii) The CTW algorithm produces good speedup for symmetric workloads, and in particular when size of a network, message population, and the amount of computation time needed to process each event is large. Some variations in the service time of messages affect the speedup, but its impact, especially for feed-forward networks, is generally not significant. With moderate asymmetric workload (i.e., low percentages of hot-spot messages for MIN, or few lazy nodes in Torus), the speedup is not changed radically. However, rapid drop in the speedup occurs with large amount of asymmetry.

At the same time the experiments revealed several limitations of the CTW-algorithm. First, it requires that size of application is much larger than size of hardware. For instance simulating a 4x4 torus (16 objects) on a Sequent Symmetry with 16 processors produces poor performance. However, increasing the size of the network to 20x20 leads to substantially better speedup. Second, the CTW-algorithm performs poorly for heterogeneous applications. This kind of applications were studied by introducing lazy nodes in the torus net (Figure 17), and assuming hot-spots in MINs (Figure 12). This phenomenon can also be observed when variation of the $d_{ij}$ for different objects is very high.

## ACKNOWLEDGEMENTS

The paper was reworked and rewritten as the first author was spending a year at the Georgia Institute of Technology. The use of the resources at Georgia Tech and the valuable supports given by Richard. M. Fujimoto are greatefully acknowledged. This research has been supported by the Swedish National Board for Technical Development (STU).

## REFERENCES

Ayani, A., 1991. Parallel Discrete-Event Simulation on Shared Memory Multiprocessors. *International Journal in Computer Simulation.* 1:81 - 97.

Chandy, K. M. and Misra, J. 1979. Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. *IEEE Transactions on Software Engineering,* SE-5:440-452.

Fujimoto, R.M., 1989. Time Warp on Shared Memory Multiprocessor. *Transactions of the Society for Computer Simulation.* 6:211-239.

Jefferson, D.R., 1985. Virtual Time. *ACM Transactions on Prog Lang. & Syst.* 7:77-93.

Lubachevsky, B.D., 1988. Bounded lag distributed discrete event simulation, *in Proc. of the SCS Western Multiconference on Distributed Simulation,* pp. 183-191.

Lubachevsky, B.D., 1989. Efficient Distributed Event-Driven Simulations of Multiple-Loop Networks. *Communications of the ACM.* 32:111-131.

Misra, J., 1986. Distributed Discrete-Event Simulation. *ACM Comp. surveys.* 18:39-65.

Nicol, D., 1988. Parallel Discrete Event Simulation of FCFS Stochastic Queueing Networks. *Parallel Programming: Experience with Applications, Languages and Systems, ACM-SIGPLAN,* 124-137, July.

Nicol, D., 1991. Performance Bounds on Parallel Self-Initiating Discrete-Event Simulations. *ACM Trans. on Modeling and Computer Simulation,* Vol. 1, No.1, 24-50.

Peacock J.K., Wong J.W., and Manning E.G., 1979. Distributed Simulation Using a Network of Processors. *Computer Networks,* vol. 3:44 - 56.

Rajaei, H., 1992. SIMA: An Environment for parallel Discrete Event Simulation. *Proceedings of the 25th Annual Simulation Symp.* 147-155.

Reed D.A., Malony A.D., and B. D. McCredie B.D., 1989. Parallel Discrete Event Simulation using Shared Memory. *IEEE transactions on software Eng.* 14:541-553.

Righter, H., and Walrand J.C., 1989. Distributed Simulation of Discrete Event Systems. *Proceedings of the IEEE,* 77:99-113.

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 Western Multiconference on Distributed Simulation,* 34-42.

Wagner D.B., Lazowska E.D., and Bershad B.H., 1989. Techniques for Efficient Shared-Memory Parallel Simulation. *in Proc. of the SCS Eastern Multiconference on Distributed Simulation.* 29-37.

## AUTHOR BIOGRAPIES

**RASSUL AYANI** received his D.I degree from Technische Hochshule in Vienna, Austria (1970) and his Ph.D degree from the Royal Institute of Technology in Stockholm. He is an Associated Professor in the Departement of Telecommunications and Computer System, Royal Institute of Technology in Stockholm, Sweden. His current research interests are in parallel architectures, parallel algorithms and parallel simulation. He is Associate Editor of the ACM Transactions on Modeling and Computer Simulation (TOMACS) and a member of ACM, IEEE and SCS.

**HASSAN RAJAEI** received a MS from U. of Utah in 1979. He has been at the Royal Institute of Technology in Stockholm since 1985 and currently is in the process of finishing his Ph.D. His research interests are parallel simulation, communication systems, and distributed systems. He is a member of SCS.