# Exploiting Temporal Uncertainty in Parallel and Distributed Simulations

Richard M. Fujimoto
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280

**Abstract**

*Parallel discrete event simulation algorithms are usually based on time stamp ordering of events. Distributed virtual environment (DVE) applications such as DIS typically use unordered event delivery. A partial order called approximate time (AT) is proposed to order events in both domains, facilitating reuse of simulations across DVE and analysis applications. A variation on AT-order called approximate time causal (ATC) order is also described. Synchronization algorithms to realize these orderings are presented as well as performance measurements on a workstation cluster. A long-term goal of this work is to use AT and ATC order to exploit temporal uncertainty in the model to achieve efficient conservative parallel simulation despite little or no lookahead, a longstanding problem in the field.*

## 1 Introduction

Event ordering and performance requirements for parallel and distributed simulations depend on their intended use. Simulations for analytic applications usually rely on time stamp order to ensure the model correctly reproduces before-and-after relationships. On the other hand, unordered delivery and event processing is usually sufficient in virtual environments, in part because errors that cannot be perceived by human participants can often be ignored. However, real-time performance is critical in this domain.

Recent efforts such as the High Level Architecture [1] are intended to foster interoperability and reuse of simulations. But, different approaches to event ordering impede reuse of simulations across domains. For example, embedding time stamp ordered simulations in a DVE with unordered delivery may cause one or more simulations to fail because unanticipated orders (e.g., non-causal ordering of events) may occur. Requiring the DVE to use time stamp order may incur overheads that prevent real-time performance. Some means is required for modelers to relax order constraints to achieve adequate performance, but without allowing arbitrary event orders.

A related problem facing practitioners today concerns achieving efficient execution of parallel simulations with little or no lookahead, especially in federated simulation systems. It is well known that conservative algorithms perform poorly if the application has little lookahead [2]. Re-coding the simulation to contain large lookahead is in general difficult, and often results in fragile, hard-to-maintain code. Optimistic synchronization provides a potential solution, but also requires substantial re-engineering of the simulator to introduce rollback mechanisms. We attack this problem by exploiting temporal uncertainty in the model to achieve efficient conservative execution, despite little or no lookahead.

Here, two partial orders for events, called approximate time (AT) and approximate time causal (ATC) order are proposed. The next section reviews related work. The AT and ATC partial orders are then described, as well as distributed algorithms to realize them. Experimental results are presented to evaluate the impact of relaxing ordering constraints in zero lookahead simulations both in terms of execution speed and accuracy of the simulation.

## 2 Related Work

Relaxed order constraints for parallel simulations have been studied previously, e.g., see [3-6]. These techniques usually take a "protocol centric" point-of-view where violations of time stamp order are allowed during the execution. We focus on defining new, application independent order semantics to replace time stamp order. A fundamental assumption in our work is the modeler must have detailed control over acceptable event orders that can arise during the execution.

Time intervals (used here) and fuzzy logic have been studied as a means to specify uncertainty in simulations, e.g., see [7, 8]. This work does not address issues concerning parallel/distributed execution, but is complementary in that it gives an approach to defining time intervals. Wieland also proposes using time intervals to address issues concerning simultaneous events [9].

Work in the distributed computing community has focused on causal order for distributed computations e.g., see [10, 11]. However, this work does not consider the requirements of simulation applications. For example, simulations often use time stamps to order actions, e.g., air traffic might be arranged according to pre-defined flight schedules. Causal order does not guarantee proper ordering of events where time stamps are used to specify the desired order.

## 3 Approximate Time Order Simulations

Time stamp order assumes each event is assigned a precise time stamp, defining a total ordering of events. In reality, however, there is almost always uncertainty concerning when an event occurs. For example, the time a moving vehicle comes into view depends on how fast it is traveling, visibility conditions, etc., factors that cannot be known with complete certainty. Temporal uncertainty is ubiquitous because a simulation is only an approximation of the real world. Thus, there is no *a priori* reason to believe there is only one correct ordering of events that is acceptable during the execution of the model. Different orders can sometimes lead to the same results. One can exploit this fact to yield more efficient synchronization protocols. Further, less precise temporal specifications may suffice in virtual environments compared to analytic applications, enabling one to relax ordering constraints further, e.g., to achieve real-time performance.

### *3.1 Approximate Time Order*

Here, simulation time intervals are used to specify temporal uncertainty. Specifically, each event X is assigned a simulation time interval [E(X), L(X)] where $E(X) \leq L(X)$. E(X) denotes the earliest simulation time that the event may occur, and L(X) the latest.

AT order defines a partial ordering based on the *is-before* relation (~>). Consider two events X and Y with time intervals [E(X),L(X)] and [E(Y),L(Y)] respectively. The is-before relationship is defined as follows:

- *Non-overlapping time intervals:* if $L(X) < E(Y)$, then X ~> Y (read X *is before* Y or X *precedes* Y). For example, in Figure 1(b) A ~> E.
- *Overlapping time intervals:* if the two time intervals have at least one point in common, then no ordering relationship exists between these events. In this case, the events are said to be *concurrent*, represented X || Y. In Figure 1(b) A || B.

The ~> relationship is transitive, i.e., if X~>Y, and Y~>Z, then X~>Z. The || relationship is *not* transitive. In Figure 1(b) D||E and E||F, but D and F are *not* concurrent.

Approximate time order is identical to time stamp order if E(X) is equal to L(X) for each event. If L(X) is equal to infinity for each event, then no ordering constraints apply because all events are concurrent. Thus, time stamp order and unordered events are trivial special cases of AT-order.

Time intervals can increase the concurrency in the simulation. Figure 1(a) shows a snapshot of a simulation using time stamp order. The events shown in this figure have different time stamps, so they cannot, a priori, be processed concurrently by a conservative protocol. This is because A could affect B, which could affect C, etc. On the other hand, if the events are assigned time intervals as shown in Figure 1(b), A, B, C and D are concurrent and can be processed simultaneously on different processors. ELT and LET in Figure 1(b) will be defined later.

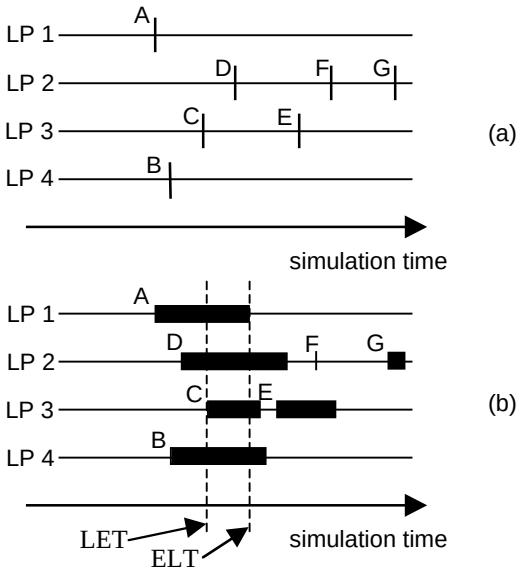


**Figure 1. Example illustrating time intervals and AT-order. (a) events using time stamp order. (b) events using time intervals.**

It is sometimes convenient to assign a precise time stamp to an event specified using a time interval. In particular, when AT-order is used to federate existing time stamp ordered simulations, such simulators expect events to be assigned precise time stamps. If each event is assigned a precise time stamp that lies within the time stamp interval for that event, the resulting event ordering will always be consistent with AT-order, i.e., if X ~> Y, then TS(X) < TS(Y) where TS(E) is the time stamp assigned to event E.

### *3.2 AT Simulations*

We assume the parallel/distributed simulation consists of a collection of logical processes (LPs) that communicate exclusively by exchanging messages. An AT-order simulation utilizes the following principles:

- Each LP processes events in AT-order. For any two events X and Y residing within a single LP, if X~>Y, then X must be processed before Y. If X || Y, then X and Y may be processed in any order.
- A simulation time clock is defined. The current simulation time of each LP is defined as an interval $[C_E,C_L]$, indicating the LP's current time is greater than or equal to $C_E$, but no larger than $C_L$. In a conservative execution, the E-time of the clock never decreases. This ensures that if $C_i$ and $C_{i+1}$ are successive clock values, either $C_i \parallel C_{i+1}$, or $C_i \sim> C_{i+1}$. An event X whose time stamp interval overlaps with $C=[C_E,C_L]$ is said to be concurrent with C (X||C).
- If an LP's current simulation time is $C=[C_E,C_L]$ and that LP schedules a new event X, then $E(X) \geq C_E$, i.e., X || C or C ~> X. This rule is called the *AT-scheduling constraint.*

Lookahead constraints can be applied to AT-order simulations. Specifically, if the lookahead of an LP is L, then each event X generated by the LP at time $[C_E, C_L]$ adheres to the constraint $E(X) \geq C_E + L$. Large lookahead is not a prerequisite to achieving concurrency in AT-order simulations, but can be exploited when it exists. The focus of this work is on zero lookahead simulations, but lookahead is included in the discussions for completeness.

### *3.3 AT-Order Synchronization Algorithms*

The synchronization algorithm must ensure events are processed in AT-order and no event is delivered to an LP in its past, i.e., no event X is delivered with X~>C where C is the LP's clock. Two synchronization algorithms are described. The first is a simple, synchronous algorithm that does not attempt to exploit lookahead. The second is a more sophisticated, asynchronous algorithm that can exploit lookahead and minimum time stamp intervals.

#### 3.3.1 A Simple, Synchronous Algorithm

The first algorithm (see Figure 2) repeatedly identifies the *earliest concurrent event set* $S_E$, and then processes the events in $S_E$. If S is the set of unprocessed events across all the LPs, $S_E$ is defined as the largest set of events such that for any two events $X,Y \in S_E$, X||Y, and there is no event Z such that $Z \notin S_E$ and Z~>X (where $X \in S_E$). In Figure 1(b) $S_E$ in the first iteration includes A, B, C, and D. After processing these events, the second iteration would process E and F, and the third iteration G, assuming no new events are created.

The algorithm uses a barrier to ensure events in the current iteration have been processed before advancing to the next one. The barrier must ensure there are no transient messages in the network before releasing any LP. This can be accomplished using counters to keep track of the number of messages sent and received.

The algorithm identifies the earliest concurrent event set by computing the global minimum among the L-times of all unprocessed events. This minimum value is referred to as the *earliest L-time* or ELT. This value is shown in Figure 1(b), where it can be seen the events in $S_E$, A, B, C, and D, are the four events that overlap with ELT. The following lemma verifies that the set of events whose time interval overlap with ELT are concurrent events, and no events in the set of pending events precedes these events in approximate time order.

**Lemma 1**. *Consider a non-empty set of events S. Let ELT be the minimum L-time among the events in S. Let $S_E$ be the set of events {X: E(X) ≤ ELT ≤ L(X)}. Then for any event $X \in S_E$, there is no event $Z \in S$, such that Z ~> X, and for any events $X,Y \in S_E$, X || Y.*

Proofs of this and the lemmas and theorems that follow are presented in [12].

After determining $S_E$, a second global value, the latest E-time (LET) among the events in $S_E$, is computed. This value is also shown in Figure 1(b). The LET value is used to set the E-time of the simulation clock in each LP. The clock interval [LET,ELT] represents the intersection of the time intervals of events in $S_E$. The next lemma thus becomes apparent.

```
Clock [C_E,C_L] = [0,0]
while (S ≠ ∅)
  ELT = min (L(X)) for all X ∈ S
  S_E = {all events Y: E(Y)≤ELT≤L(Y)}
  LET = max (E(Z)) for all Z ∈ S_E
  Clock [C_E,C_L] = [LET, ELT]
  process events in S_E
  barrier
end-while
```

**Figure 2. Algorithm 1: a simple AT-order simulation algorithm.**

**Lemma 2.** *Consider a non-empty set of events S. Let ELT be the minimum L-time among the events in S, and $S_E$ be the set of events {X: E(X) ≤ ELT ≤ L(X)}. Let LET be the maximum E-time among the events in $S_E$. Then all points in the interval [LET, ELT] must be included in the interval for each event in $S_E$.*

The following lemma verifies that the clock values produced by algorithm 1 form a non-decreasing sequence.

**Lemma 3.** *The sequence of E-time values stored into the simulation clock defined in algorithm 1 form a non-decreasing sequence.*

We observe that at the end of each iteration of algorithm 1, the E-time of the simulation clock is equal to the maximum E-time among all events that have been processed thus far in the parallel simulation. The following theorem verifies the correctness of algorithm 1, i.e., the algorithm does process events in AT-order, and no LP processes an event in its past.

**Theorem 1.** Algorithm 1 guarantees that no LP will process events out of AT-order. Further, no LP will process an event X such that X~>C, where C is the LP's clock value when the event is processed.

#### 3.3.2 An Asynchronous AT-Order Algorithm

The principal virtue of algorithm 1 is its simplicity. This algorithm can be improved in several ways, however:

- Each iteration requires two reduction computations to compute ELT and LET. The LET computation can be eliminated by setting the simulation clock to ELT rather than [LET, ELT].
- The algorithm does not attempt to exploit lookahead. or minimum time interval size.

- An asynchronous version of the algorithm is preferable to reduce the time LPs spend waiting for others to reach the synchronization point.

The AT-order synchronization algorithm must ensure events are processed in AT-order, and an LP does not receive an event with time interval that precedes the LP's current time. These objectives can be met by computing the following value for each LP:

**Lower Bound on Late-Time (LBLT).** The $LBLT_i$ for $LP_i$ is a lower bound on the L-time value of any future message that could later be received by $LP_i$.

LBLT differs from ELT because ELT is the minimum L-time of any event in a snapshot of the simulation, but LBLT is a lower bound on the L-time of future messages that may be received. The LBLT value for an LP is similar to the lower bound on the time stamp (LBTS) of messages an LP may later receive in a time stamp order parallel simulation. Specifically, the synchronization protocol can guarantee that $LP_i$ does not receive messages in its past by ensuring $C_E \leq LBLT_i$, where $C_E$ is the E-time of the LP's simulation clock. Similarly, one can guarantee the LP processes events in AT-order by requiring that $E(X) \leq LBLT_i$ for any event X processed by the LP.

Existing time stamp order synchronization algorithms can be adapted to compute LBLT. One such algorithm is described next. This algorithm is not unlike conservative algorithms such as [13]. It is assumed any LP can send a message to any other. A single lookahead value is associated with $LP_i$, denoted $L_i$. Also, it is assumed all messages sent by $LP_i$ have a minimum time interval size of at least $MTSI_i$, i.e., $L(X)-E(X) \geq MTSI_i$ for each message X sent by the $LP_i$. Either or both of $L_i$ and $MTSI_i$ may be zero. This algorithm is asynchronous, i.e., LBLT computations proceed in background, as needed, and no global barriers are required. In the following, the $LP_i$'s simulation time clock is denoted $[CE_i, CL_i]$.

$LBLT_i$ values are computed using the following values:

- $LTOut_i$: a lower bound on the L-time of future outgoing messages $LP_i$ may generate. $LTOut_i$ is computed as $CE_i+L_i+MTSI_i$. This value is used in the LBLT computation if the LP is *not* blocked.
- $CLTOut_i$: a conditional lower bound on the L-time of future outgoing messages $LP_i$ may generate, assuming no additional messages are passed to $LP_i$ from other LPs. Typically, this value will be computed as $min(E(X))+L_i+MTSI_i$ where $min(E(X))$ is the minimum E-time among unprocessed events in $LP_i$. The larger of $LTOut_i$ and $CLTOut_i$ is used in the LBLT computation if the LP is blocked, waiting for LBLT to advance.
- L(X) for transient messages X: the L-time of messages that have been sent, but have not been received while the LBLT computation is being performed.

Using these values, $LBLT_i$ can be computed as the minimum along a consistent cut of the computation [14].

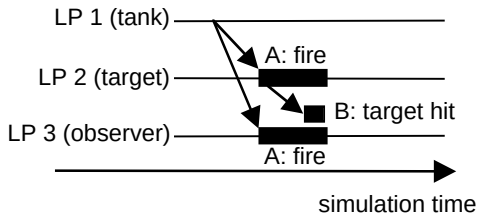


**Figure 3. Ordering of causally related events.**

Transient messages are those that cross the cut. This computation is similar to that used to compute GVT in optimistic simulations. $LBLT_i$ gives the simulation executive sufficient information to deliver events to the LP and to advance its clock.

## 4 Approximate Time Causal Order

One limitation of AT-order is it may not correctly order causal relationships. Consider the scenario depicted in Figure 3. $LP_1$ is modeling a tank that schedules an event to indicate it is firing upon a target. After receiving this event, $LP_2$, modeling the target, schedules a new event indicating it has been hit. Both events are sent to a third, observer LP. If the "fire" and "hit" events are assigned overlapping time intervals, an AT-order simulation may process them in any order. This is unfortunate. The fire event must precede the hit event because there is a causal relationship between the two. While one could force a particular order by assigning appropriate, non-overlapping time intervals, this can become cumbersome, particularly if causal relationships are spread over LPs on different processors.

### *4.1 Event Ordering*

An alternate solution is to strengthen AT-order to ensure proper ordering of causal relationships. For this purpose we define AT-causal (ATC) order. ATC order is similar to AT ordering, except causal order is imposed on the order of events containing overlapping time intervals.

Causal ordering is based on Lamport's happens before ($\rightarrow$) relationship [15]. The computation in each logical process is viewed as a sequence of actions. Here, an action is a computation that processes an event, sends a message, or receives a message. The happens before relationship is defined as follows:

- If X and Y are two actions within the same LP and X precedes Y, then $X \rightarrow Y$ (read X happens before Y).
- If X is sending a message M and Y is receiving M, then $X \rightarrow Y$.
- If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$ (transitivity).

As before, each event X is assigned a time interval [E(X), L(X)]. AT-causal order (represented $\sim>_C$) is defined as follows:

- If for two events X and Y $L(X) < E(Y)$, then $X \sim>_C Y$.
- If X and Y have at least one time point in common, and $X \rightarrow Y$, then $X \sim>_C Y$.

- If neither of the above conditions apply, then X and Y are concurrent events (X || Y).

For example, in Figure 3, $A \sim>_C B$ because $A \rightarrow B$ and the two events have overlapping time intervals.

### *4.2 ATC Simulations and Algorithms*

ATC simulations follow the same principles as AT simulations except ATC is used as the ordering mechanism rather than AT order. Specifically, each LP in an ATC simulation must process events in ATC order. Further, an LP should never receive an event with a time interval preceding that of the LP's current time interval.

It is clear that the principal new requirement that must be satisfied to realize an ATC simulation compared to an AT simulation is ensuring that concurrent AT-order events are processed in causal order. Many approaches have been developed to realize causal order. A simple technique is to use global synchronization points. Specifically, causal ordering can be guaranteed by forcing all concurrent events to be processed before any of the new events produced as a result of processing these events are delivered to LPs. Algorithm 1 described earlier for implementing AT-order simulations can thus be used to realize ATC simulations provided one ensures successive concurrent messages sent by one LP to another are delivered in the same order in which they were sent. The barrier synchronization at the end of each iteration ensures causal ordering of concurrent events. For example, in Figure 3 both instances of the fire event will be processed in one iteration, and the hit event will be processed in a later iteration.

## 5 Experimental Evaluation

An implementation of the AT-order algorithm was developed. The principal questions investigated in this study concern the performance benefits that can be achieved by introducing time intervals, and the impact of time intervals on the accuracy of the simulation results that are produced.

### *5.1 The Approximate Time RTI*

An important application of AT and ATC order is in developing efficient federations of simulations, e.g., distributed simulation systems realizing the High Level Architecture. In particular, a goal of these initial performance studies was to evaluate the effectiveness of AT-order compared to time stamp order execution in federating simulations with zero lookahead. Each LP is a sequential simulator augmented to interface it to the distributed simulation system. Here, we adopt HLA terminology; each simulator (LP) is called a federate, and the parallel simulation executive is called the Runtime Infrastructure (RTI).

Internally, each federate implements a traditional event-oriented execution mechanism using *precise* time stamps. Approximate time ordering is only used for messages transmitted between federates. The federate assigns a time interval to each event sent to other federates, but local events that remain within the federate are only assigned a precise time stamp. Further, incoming messages are assigned a precise time stamp that lies within the message's time interval before they are passed to the federate. Each federate then process all events, both locally and remotely generated, in time stamp order. The clock for each federate is defined as a precise time value.

The Approximate Time RTI (AT-RTI) was developed as the distributed simulation executive for this work. This RTI implements a portion of the interface specification for the High Level Architecture [1], modified to use an AT-order message delivery service.

In addition to assigning a time interval to each event, the sender also assigns a *target time stamp*. This value indicates the time stamp the federate would have assigned had precise time stamps been used. This information is used when a time stamp is later assigned to the event by the receiving federate. Specifically, before each message is delivered to the federate, the AT-RTI makes a call back to the federate specifying a range of *allowed time values* that may be assigned to the event (a subset of the time interval assigned by the sender) and the target time stamp. The federate returns the precise time value that it wishes to assign. The AT-RTI then delivers events to the federate in time stamp order. In the experiments described here the federate uses the target time stamp if it is within the allowed time interval, and the closest available time stamp if it is not.

The asynchronous version of the AT-order synchronization algorithm described earlier is used in this implementation. A distributed snapshot algorithm similar to Mattern's GVT algorithm [14] is used to compute LBLT values. This algorithm and its implementation are described elsewhere [16]. Key features of this algorithm include:

- The algorithm is asynchronous, and executes "in background" concurrent with other simulation computations, and does not require barriers.
- The algorithm computes a global minimum along a consistent cut of the distributed computation.
- A butterfly reduction network is used to compute global minimum values, allowing a global minimum to be computed and distributed to each of the N processors in log N steps, in the absence of transient messages.
- A coloring scheme is used to detect transient messages.
- Counters of the number of sent and received messages are used to determine when all transient messages have been received, and have been accounted for in the global minimum computation.

The AT-RTI on each processor initiates a new LBLT computation whenever the federate on that processor requests its simulation time be advanced (via a service similar to the HLA Next Event Request service), but the request cannot be granted and no events are available for

delivery to the federate. If more than one processor simultaneously initiates a new LBLT computation, the AT-RTI software automatically combines these initiations into one LBLT computation. In the experiments described here, a processor will not initiate a new LBLT computation if one is already in progress, although the LBLT software supports multiple, concurrent LBLT computations to be in progress at one time. If the LBLT computation does not return a value that is sufficiently large to complete the federate's request to advance simulation time, a new LBLT computation is initiated. The federate and RTI are free to perform other computations, e.g., new messages may arrive, while an LBLT computation is in progress.

### *5.2 Cluster Environment*

A cluster of eight Sun Ultrasparc-1 workstations (Solaris 5.5) interconnected by a low latency Myrinet switch using LANai Version 4.1 Myrinet cards with 640 MBit/second links was used. Newer 1.28 GBit switches are currently available, but were not used here. Myrinet switches provide point-to-point interconnection between workstations, and feature cut-through message routing to reduce message latency [17].

The AT-RTI software was developed using the RTI-Kit software package [16]. RTI-Kit is a collection of libraries implementing key mechanisms that are required to realize distributed simulation RTIs. The AT-RTI uses RTI-Kit's MCAST library that provides group communications facilities, and the TM-Kit library that implements time management functions. The version of RTI-Kit used in this study is built over version 2.0 of the Fast Message software package developed at the University of Illinois [18]. Other implementations of RTI-Kit are built over TCP/IP or shared memory, but were not used in this study.

Federate-to-federate communication latency using the RTI-Kit software was measured at 20 to 30 microseconds for messages of 128 bytes or less on the Sun/Myrinet configuration. A global reduction operation requires about 20 microseconds for two processors, and 70 microseconds for eight.

### *5.3 Applications*

Here, we are particularly interested in simulations with zero lookahead. In these initial studies the PHOLD synthetic workload [19] and a queueing network were used, as these are standard benchmarks used in the PDES community. Other experiments with battlefield simulations are currently in progress. In PHOLD, a single LP is mapped to each federate, and LPs do not send messages to themselves. Thus, there are no local events, i.e., each event generates a message that is sent to a different processor. The target time stamp is selected from an exponential distribution with mean 1.0, and the destination federate is selected from a uniform distribution. The minimum time stamp increment is zero, resulting in a zero lookahead simulation.

The second application is a queueing network, configured in a toroid topology. This simulation does *not* attempt to exploit lookahead. A textbook approach to realizing this simulation was used that includes both job arrival and job departure events. Each departure event schedules an arrival event at another queue with time stamp equal to the current simulation time of the LP scheduling the event, i.e., messages sent between LPs all contain zero lookahead. Service times are selected from an exponential distribution, and jobs are serviced in first-come-first-serve order. Each federate models a rectangular portion of the queueing network.

It is well known that certain classes of queueing network simulations can be optimized to exploit properties of the queues to improve their lookahead. This approach was intentionally *not* used here, since such techniques are generally difficult to apply in large, complex models. The experiments performed here were intended to evaluate the effectiveness of AT-order in federating simulations without rewriting the models to exploit lookahead.

### *5.4 Execution Speed*

A sequence of experiments was performed using AT-order where the interval size was varied. Each experiment was repeated five times, using different random number generator seeds in each execution. All messages sent between federates use the same interval size, which was also used as the MTSI value. As mentioned earlier, all simulations have zero lookahead. The data point with zero interval size corresponds to the case where precise time stamps and time stamp order processing are used.

PHOLD uses one LP per federate, and one federate on each of the eight processors. Job populations of 256 and 4096 were used. The queueing network simulations modeled 256 (16 by 16) and 4096 (64 by 64) queues, and in each case assumed fixed populations of 256 and 4096 jobs, respectively, or one job per queue.

The performance of the simulation is plotted relative to the distributed simulation using precise time stamps in Figure 4. Increases in the interval size lead to significant improvements in performance, more than a twenty-fold reduction in execution time in some cases. Even modest sized time intervals lead to significant performance improvements. For example, an interval size of 0.1 for the PHOLD application implies the time stamp increment is set to $X \pm 0.05$, or $\pm$ 5% in most cases since the average time stamp increment is 1.0. A conservative time stamp ordered simulation using the same distributed snapshot algorithm used in the AT-RTI was also implemented. The performance of this implementation is only slightly better (approximately 5%) than the AT-RTI using time intervals defined as a single point.

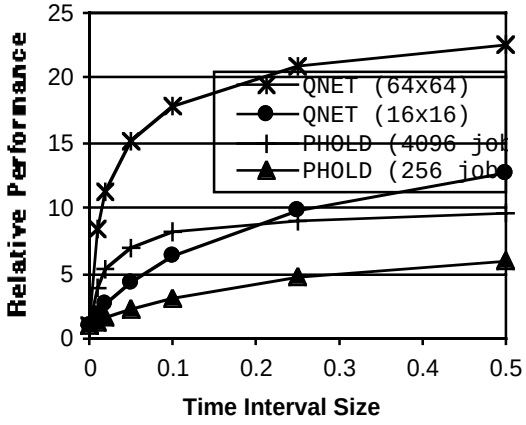


**Figure 4. Execution Speed using AT-Order.**

The performance improvement can be attributed to reduced synchronization overhead resulting from the use of time intervals. In particular, because these simulations contain zero lookahead, a synchronization step is required after virtually every event in a time stamp ordered simulation. Time stamp intervals allow events occurring at approximately the same time to be processed concurrently, greatly reducing the number of synchronization operations. This is the principal reason for the improved performance.

The data in Figure 4 also indicates greater improvement occurs for larger simulation models. This is because for large models, there are more events that have overlapping intervals, leading to increased concurrency. In general, AT order can be expected to yield greater concurrency as the number of pending events increases.

### *5.5 Accuracy*

The PHOLD and queueing network simulation computations were examined in order to assess the impact of using time intervals on the results computed by the simulation. In PHOLD, the time interval is selected so that the target time stamp is in the middle of the interval, except in cases where this would cause the beginning of the interval to precede the current time of the federate. In this case, the interval was shifted to begin at the federate's current logical time. Recall the federate's current time was defined as a precise time value in these experiments.

The RTI was instrumented to count the number of times the assigned time stamp exactly matched the target time stamp. The fraction of times the target time stamp was assigned to the event is shown in Figure 5(a). For interval sizes up to 0.1 (± 0.05) over 98% of the assigned time stamps hit the target, and the average error was less than 0.0003 (see Figure 5(b). Even with intervals as large as 0.5 (± 0.25) almost 90% (or more) of the assignments hit the target, and the average error was less than 1%.

The queueing network simulations computed two statistics: the average waiting time encountered by a job when it arrived at a station, and the average number of jobs served in the network per unit of simulation time. Note these are statistics computed by the simulation, as opposed to parallel performance statistics. Here, simulation results could be compared against the exact solution because these queueing networks have simple closed form solutions. Data for the larger (64 by 64) queueing network is shown in Figure 6. Figure 6(a) shows the job waiting time and Figure 6(b) the job throughput of the network as the job population, and thus the amount of queueing, increased. The exact solution, simulation using precise time stamps, and approximate time simulations are shown. The simulator using precise time stamps was always within 5% of the exact solution. The approximate time simulations were within 1% of the precise time stamp result (and 5% of the exact solution). These results indicate that the use of relaxed ordering has virtually no impact on the numerical results computed by the simulation for the models and statistics that were tested.

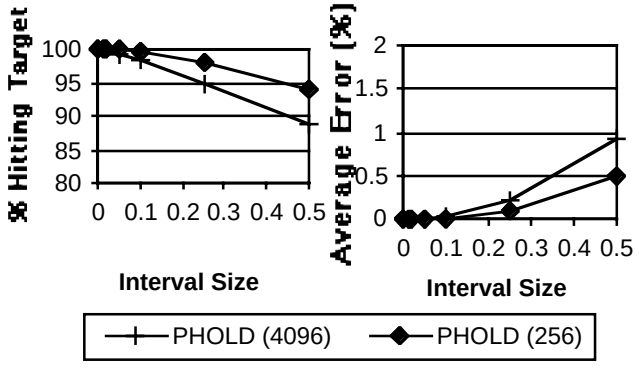


**Figure 5. PHOLD accuracy. (a) fraction of assigned time stamps hitting taget. (b) Average time stamp error.**

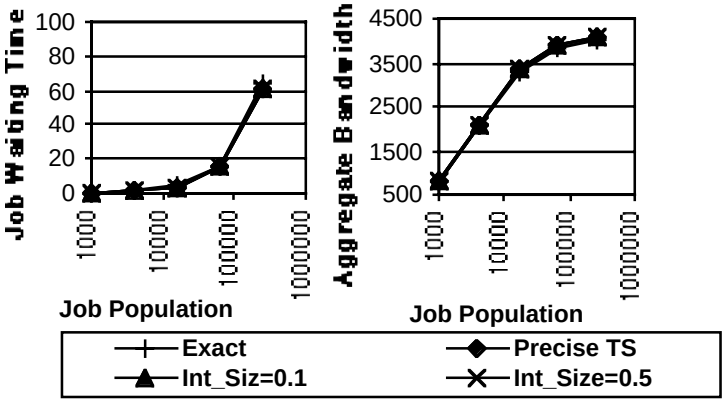


**Figure 6. Accuracy of the queueing network simulation. (a) mean job waiting time. (b) jobs processed per unit simulation time.**

## 6 Conclusions and Future Work

Approximate time and approximate time causal order provide an alternate approach to parallel/distributed simulation. Because AT-order includes time stamp order and unordered communications as special cases, it provides an approach that spans analytic simulations and virtual environments. Initial experiments indicate that this approach shows promise in achieving good parallel performance of zero lookahead simulations without resorting to optimistic processing techniques. The performance gains were realized with negligible impact on the numerical results produced by the simulator for the test applications that were examined.

The results described here only scratch the surface in the use of AT and ATC order. Numerous issues remain to be investigated. A few are listed below.

- Model validation. What size time intervals can be used without invalidating the simulation model?
- Application studies. More extensive investigation of the use of these ordering mechanisms in real world applications is required, especially to evaluate its impact on computed results.
- Repeatability. Additional mechanisms are required to ensure repeated executions of the simulation with the same input and initial state produce the same results.
- Time stamp assignment. In simulations requiring precise time stamps (e.g., federating conventional simulations) more sophisticated techniques to assign time stamps may be used. An approach where time stamps are assigned according to a probability distribution is under investigation. Another problem is the case where there are multiple recipients of messages for the same event. It may be necessary to ensure the same time stamp is assigned to each message.
- Synchronization algorithms. More advanced algorithms can improve the efficiency of the AT and ATC ordering mechanisms. In particular, algorithms that exploit topology have not yet been examined.
- Use in wide area networks. Approximate time synchronization provides a means to reduce synchronization overheads. Such overheads are particularly onerous in wide area networks. Thus a potential application is in enabling geographically distributed simulations.

While many open questions remain, the initial results using AT and ATC order reported here are encouraging. A principal conclusion of this preliminary work is time intervals hold excellent promise for synchronizing parallel/distributed simulations, and merit further study.

## 7 Acknowledgements

We thank Fred Wieland, the parallel simulation groups at Georgia Tech and the Defence Evaluation and Research Agency in Malvern (where much of this work was done), and the anonymous referees for their comments. Funding was provided under Darpa contract MDA972-97-C-0017.

## 8 References

1. Defense Modeling and Simulation Office, *High Level Architecture Interface Specification, Version 1.3*, 1998: Washington D.C.
2. Fujimoto, R.M., *Performance Measurements of Distributed Simulation Strategies.* Transactions of the Society for Computer Simulation, 1989. **6**(2): p. 89-132.
3. Sokol, L.M. and B.K. Stucky, *MTW: Experimental Results for a Constrained Optimistic Scheduling Paradigm,* in *Proceedings of the SCS Multiconference on Distributed Simulation*. 1990. p. 169-173.
4. Porras, J., J. Ikonen, and J. Harju, *Applying a Modified Chandy-Misra Algorithm to the Distributed Simulation of a Cellular Network*, in *Proceedings of the 12th Workshop on Parallel and Distributed Simulation*. 1998. p. 188-195.
5. Rao, D.M., *et al., Unsynchronized Parallel Discrete Event Simulation*, in *Proceedings of the Winter Simulation Conference*. 1998. p. 1563-1570.
6. Martini, P., M. Rumekasten, and J. Tolle, *Tolerant Synchronization for Distributed Simulations of Interconnected Computer Networks*, in *Proceedings of the 11th Workshop on Parallel and Distributed Simulation*. 1997. p. 138-141.
7. Allen, J., *Maintaining Knowledge about Temporal Intervals.* Communications of the ACM, 1983. **26**(11): p. 832-843.
8. Dubois, D. and H. Prade, *Processing Fuzzy Temporal Knowledge.* IEEE Transactions on Systems, Man, and Cybernetics, 1989. **19**(4): p. 729-744.
9. Wieland, F., *The Threshold of Event Simultaneity*, in *Proceedings of the 11th Workshop on Parallel and Distributed Simulation*. 1997. p. 56-59.
10. Birman, K., A. Schiper, and P. Stephenson, *Lightweight Causal and Atomic Group Multicast.* ACM Transaction on Computer Systems, 1991. **9**(3): p. 272-314.
11. Raynal, M. and M. Singhal, *Logical Time: Capturing Causality in Distributed Systems.* IEEE Computer, 1996. **29**(2): p. 49-56.
12. Fujimoto, R.M., *Exploiting Temporal Uncertainty in Parallel and Distributed Simulations*, 1998, Technical Report, Georgia Institute of Technology: Atlanta, GA.
13. Chandy, K.M. and R. Sherman, *The Conditional Event Approach to Distributed Simulation*, in *Proceedings of the SCS Multiconference on Distributed Simulation*, 1989, Society for Computer Simulation. p. 93-99.
14. Mattern, F., *Efficient Algorithms for Distributed Snapshots and Global Virtual Time Approximation.* Journal of Parallel and Distributed Computing, 1993. **18**(4): p. 423-434.
15. Lamport, L., *Time, Clocks, and the Ordering of Events in a Distributed System.* Communications of the ACM, 1978. **21**(7): p. 558-565.
16. Fujimoto, R.M. and P. Hoare, *HLA RTI Performance in High Speed LAN Environments,* in *Proceedings of the Fall Simulation Interoperability Workshop*. 1998: Orlando, FL.
17. Boden, N., *et al., Myrinet: A Gigabit Per Second Local Area Network.* IEEE Micro, 1995. **15**(1): p. 29-36.
18. Pakin, S., *et al., Fast Message (FM) 2.0 Users Documentation*, 1997, Department of Computer Science, University of Illinois: Urbana, IL.
19. Fujimoto, R.M., *Performance of Time Warp Under Synthetic Workloads*, in *Proceedings of the SCS Multiconference on Distributed Simulation*. 1990. p. 23-28.