Proceedings of the 2003 Winter Simulation Conference
S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds.

PIGGY-BACKED TIME-STEPPED SIMULATION WITH 'SUPER-STEPPING'

S.C. Tay

Centre for Remote Imaging, Sensing and Processing
National University of Singapore
Kent Ridge, 119260, SINGAPORE

G.S.H. Tan
K. Shenoy

School of Computing
National University of Singapore
Kent Ridge, 119260, SINGAPORE

ABSTRACT

We propose an optimization technique for reducing global synchronizations in traditional time-stepped simulations. Time-stepped simulations are known to be efficient when events are frequent or dense. However, when events are less frequent (when compared to the size of time-steps) the performance of time-stepped simulations drop noticeably. This paper aims at improving the performance of traditional time-stepped simulations during low frequency periods and maintaining its efficiency during high frequency periods. We focus on interactive simulations which have tight real-time interactive constraints. The proposed optimization is achieved by informing the host about future events. This information is 'piggybacked' on the ready messages sent by the participating Processing Elements (PE) to the host. We maintain simulation efficiency by switching between the proposed technique and the traditional technique depending on the observed event density. To achieve this switching we introduce a concept called 'super-stepping'. A probabilistic method is used to optimize 'super-step' size.

1 INTRODUCTION

Computer simulations are widely used by the scientific community and industry today for studying, analyzing and predicting the behavior of real-world systems. In this paper we focus on the time-stepped approach that has been commonly used to synchronize events for war game simulation on a parallel or distributed platform. In the conventional time-stepped approach, the time interval to be advanced after synchronization is performed is usually predetermined and is equal to the step-size. In the real world the density of events can be highly varying depending on the period of time being simulated. Defense simulations have marked differences in event densities during time of war (high event density combined with need for fine-granularity and real-time response) as compared to peace time (relatively low density and response time is not mission critical). This research work is aimed at optimizing the synchronization for such simulations without affecting causality constraints and without violating any real-time constraints.

During low event density intervals PEs invest processing time in scanning through the event queue at each time step even though some of the time-steps may not contain any events. The proposed technique informs PEs about a time-step only when there are events to be processed at that step. To achieve this, each PE informs the host about the future events spawned by it. Though the proposed technique reduces the number of synchronization events compared to the conventional technique, it also introduces an overhead in terms of processing time at the host and at the participating PEs. From our experiments we observe that as event density increases beyond a threshold, the traditional time-stepping algorithm performs more efficiently. As simulation is a dynamic process, an adaptive approach which switches between the techniques based on the simulation parameters seems appropriate. In order to facilitate this switching we introduce a super-stepping mechanism which partitions the simulation time into larger non-overlapping intervals.

The rest of this paper proceeds as follows. Section 2 defines some terms used in this paper. Section 3 gives an overview of time-stepped simulation and throws light on some previous work. Section 4 introduces the piggy-backing technique. Section 5 proposes the super-stepping technique and its use in technique switching. Section 6 contains our experimental results and Section 7 contains our concluding remarks.

2 DEFINITIONS

2.1 Event Density

Event Density is defined as the average number of events received by a PE per unit simulation time during an observed simulation time interval. We represent Event Den-

sity using $\delta[x, y]$, where x and y are the lower and upper bounds of the observed time interval, throughout our work. If 23 events are observed in an interval of 10 time-units then the Event Density for that PE during the observed period is 2.3.

2.2 Event Coverage

Event Coverage is the number of time-units at which the PE under observation has an event destined for it. We denote Event Coverage by $\kappa[x, y]$ where x and y are the lower and upper bounds of the observed time interval. Event coverage can also be viewed as the probability that a time-step has events to be executed. Hence, $0 \le \kappa[x, y] \le 1$. If 3 out of 10 observed time-steps had events to be executed at a particular PE then the Coverage for that PE during the observed period is 0.3.

2.3 Barrier Synchronization

In Discrete Event Simulation (DES) all events have an associated time-stamp indicating at what simulated time the event should occur. In a sequential simulation the events are processed in time stamp order to ensure that an event cannot affect its past history (if this should occur it is known as a causality error). In Parallel DES, each PE asynchronously processes events in parallel with the other PEs. Thus, instead of a centralized clock, the PE has its own logical clock, the Local Virtual Time, that runs independently of the other PEs. For this reason, the PEs have to be synchronized to avoid causality errors. A commonly used technique to achieve this synchronization is by using the barrier primitive. When a process invokes the barrier primitive, it will block until all other processors have also invoked the barrier primitive. When the last process invokes the barrier, all processes can proceed further in simulation time. Barrier Synchronization is required because the calculation of the next set of safe events assumes that there are no events in transit and that all events that were sent have been received.

3 TIME-STEPPED SIMULATION AND PREVIOUS WORK

In time-stepped simulations all participating entities in the simulation are at the same time-step at any point in wall-clock time. Typically the entire span of simulation is divided into equal-sized time steps and the simulation advances from one time step to the next. At the i^{th} step, the algorithm simulates all events that fall in the time interval $[(i - 1)\Delta, i\Delta]$, where Δ is a design parameter (Eick, Greenberg, Lubachevsky, and Weiss 1993). If Δ is very small the efficiency of the method degenerates since barrier synchronization is wasted on intervals that can contain no events. Thus, Δ is chosen large enough so that a processing element typically has several events to process in any given interval $[(i - 1)\Delta, i\Delta]$.

However, if the value of Δ is too large the simulation becomes coarse-grained as all events in the interval are simulated as if they occur simultaneously. Figure 1 illustrates the advancement of conventional time-stepped simulations.

The optimization proposed in this paper is an attempt to apply some concepts from event-driven simulations to the traditional time-stepped mechanism. In this section a brief survey of some of the relevant previous work in the field of conservative synchronization techniques is presented.

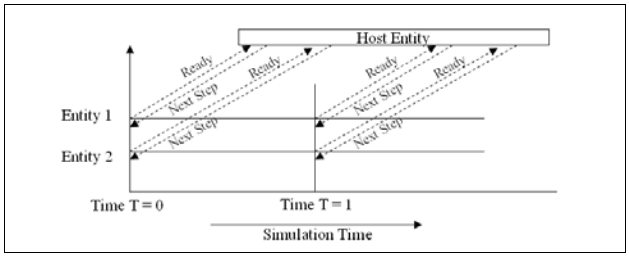

Figure 1: Simulation Progress in Traditional Time-Stepped Simulations

Conservative synchronization algorithms fall under 3 major categories depending on the approach used in handling deadlock. The first category of algorithms aims to avoid deadlocks entirely. The second category allows simulations to proceed until a deadlock is detected and then deadlock recovery algorithms (Chandy and Misra 1981) are applied to resume simulation progress. The third category is similar to the second category but the only difference is that these algorithms explicitly control the simulation progress when the entire computation stops rather than relying on the system becoming deadlocked. In order to achieve this, these algorithms rely on a mechanism called barrier synchronization.

The ideas presented in this paper were influenced by the time-of-next-event (TNE) algorithm proposed by Groselj and Tropper (1988), the windowing algorithms proposed by Lubachevsky (1988) and by Ayani (1988) and the Conservative Superstep protocol proposed by Cai, Letertre, and Turner (1999). The TNE approach deals with the case where more than one PE is mapped onto the same physical processor. It uses the greatest lower bound of the timestamps of the event messages expected to arrive next at all empty links on the PEs on that processor to unblock the PEs on its succeeding link.

Windowing algorithms proceed in three distinct phases separated by barrier synchronization (Dickens and Reynolds 1991). In the first phase, PEs determine the simulation window cooperatively such that all events with timestamps falling within the window can be executed concurrently without the possibility of any causality errors. The second phase of a windowing algorithm consists of the

concurrent execution of all events having timestamps within the window. In the third phase, events generated as a result of the processing in the second phase are passed on to other PEs. The primary difference between the various windowing algorithms is the mechanism used to determine the events that can be processed concurrently.

The Conservative Superstep Protocol (Cai, Letertre, and Turner 1999) proposed a technique wherein the simulation proceeded in a series of supersteps, where each superstep was followed by a barrier synchronization. Each superstep consisted of two phases - the computation phase and the communication phase. The communication phase involved the communication of event messages among the participating PEs and the computation phase involved the calculation of the safetime by the PEs. In this technique, the concept of supersteps is similar to that of a 'window' in which events are guaranteed to be causally safe. Several refinements to the Conservative Superstep Protocol were proposed in Gan et. al. (2001) based on the way the safetime of each PE is calculated.

4 THE 'PIGGY-BACKED' OPTIMIZATION

The traditional time-stepped technique relies on the Pending Event List (PEL) to store the events spawned (and to be executed in the future) at each PE. At each time-step the processed events are cleared from the PEL. To implement the piggy-backing technique, we introduce a new Future Time List (FTL) at the Host, in addition to the PEL that is maintained at the PEs. A simulation proceeding using the piggy-backing technique involving two PEs and a central Host will now explained in detail. Let E_1, E_3 and E_4 be the local entities residing at PE_1 and E_2 and E_5 be the ones residing at PE_2.

To initiate the simulation the Host adds a *Start* event destined to all the participating PEs in its FTL and then dispatches it to all the PEs (Figure 2).

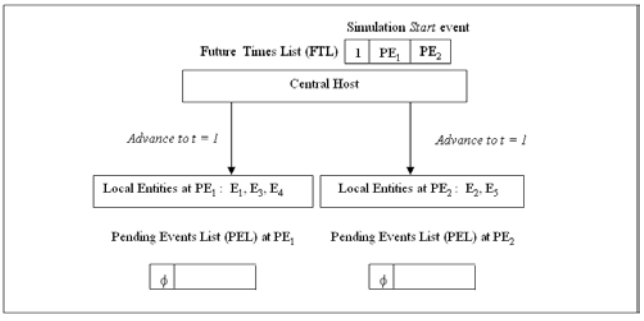

Figure 2: Initiation of the Simulation by the Host

When PEs receive the Start message from the host they scan through the PEL and execute the residing local entities. This might result in new events being generated which are meant to be processed during future times. The generated new events are added to the PEL. Figure 3 shows the PELs of PE_1 and PE_2 after the insertion of newly generated events.

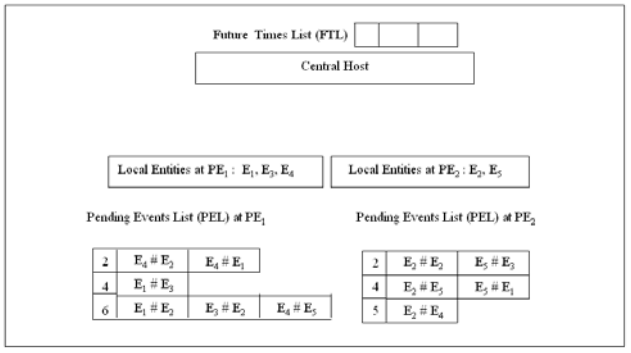

Figure 3: PELs with Newly Generated Events

Once all the local entities have been executed the PE sends the *Ready* message to the host piggy-backed with information about the new events that have been spawned as shown in Figure 4. This information is the time-stamps that destination PEs need to be informed about in order to receive *Time Advance* signals from the Host corresponding to those time-stamps.

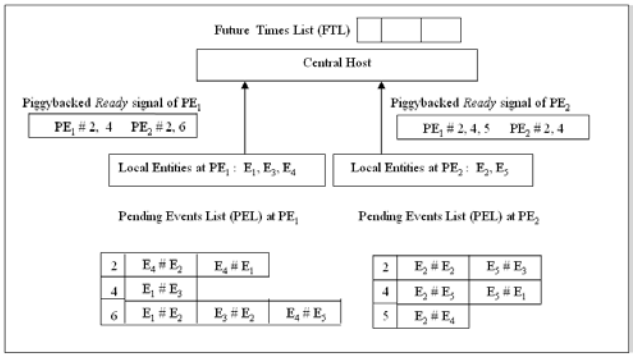

Figure 4: Piggy-Backed *Ready* Signals from the PEs

The Host extracts this piggy-backed information on receiving the *Ready* message and updates its FTL, as shown in Figure 5.

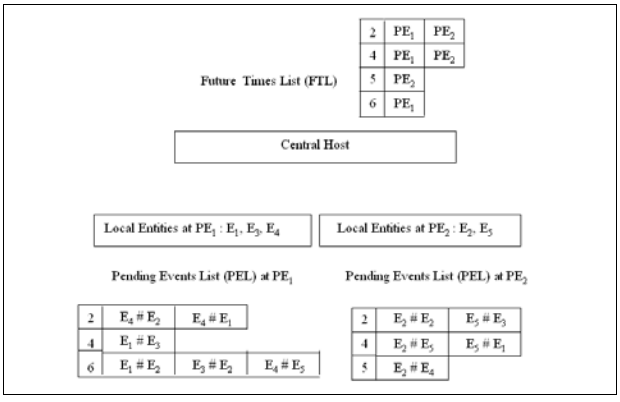

Figure 5: Updating of FTL by Host

The Host then sends *Advance to t* where *t* is the minimum time-stamp in its FTL, to those PEs corresponding to time *t* in the FTL. In this case, the Host sends an *Advance to t=2* message to PE_1 and PE_2, as shown in Figure 6. Once the PEs have been informed about the advance, the list corresponding to *t=2* in the FTL is emptied.

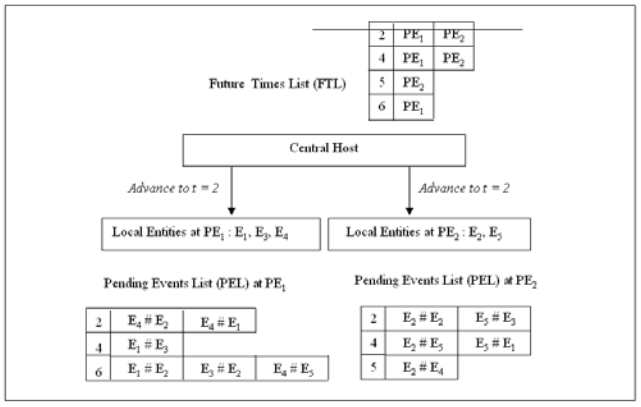

Figure 6: *Advance* Signal Being Sent to PEs

After receiving the *Advance* message from the Host, the PEs scan through the PEL, execute the residing local entities and then send the piggy-backed *Ready* signal to the Host again as shown in Figure 7 and Figure 8.

It is to be noted here that the Host only waits for *Ready* messages from those PEs that had been sent *Advance* messages during the previous time-step. When the Host receives information that it already possesses in its FTL it conveniently ignores the information. For instance, in Figure 8, the Host is already aware of the fact that PE_1 and PE_2 need to be sent an *Time Advance* signal when *t=4*. Therefore, the information that PE_1 and PE_2 need to be informed about time-stamp *4* in the new piggy-backed *Ready* signal being received by the Host is ignored. This process of the Host sending *Advance* signals and then waiting for *Ready* signals from the PEs repeats until the simulation is terminated.

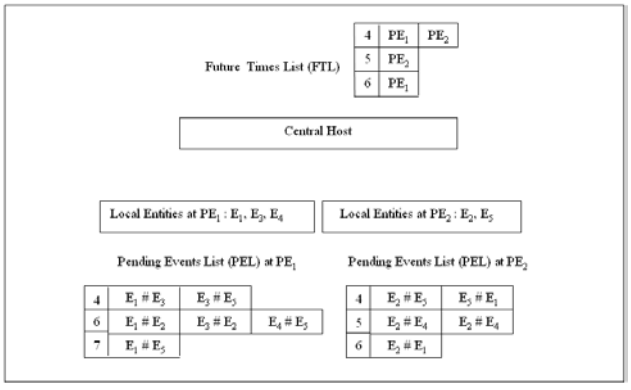

Figure 7: Flushing of PELs and Appending of New Event Information at PEs

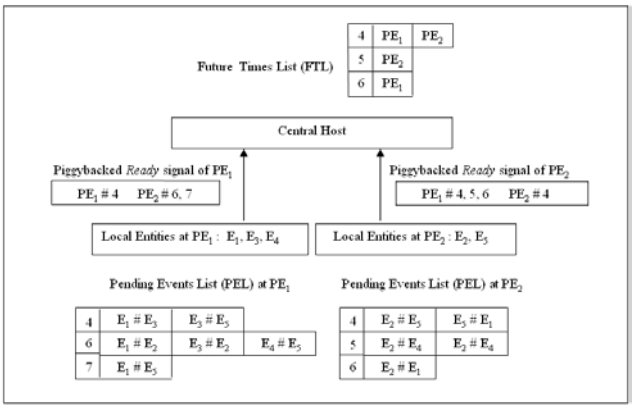

Figure 8: Sending of Piggy-Backed *Ready* Signals to Host

5 SUPER-STEPPING

Simulation is an inherently dynamic process. It is difficult to predict the behavior of any of the simulation variables reliably and accurately. Typically, an important variable to be considered is the event density. A technique that is well-suited for simulations with low event densities may prove to be inefficient when used in cases where the event density is very high.

Communication cost of inter-process messages is another factor to be considered. Communication messages are typically intra-process, LAN-based or WAN-based depending on the geographical distances between participating processes in a simulation. Techniques that avoid inter-PE communication by adding a lot of complexity at each PE may not be very useful in case the entities are being simulated by the same PE or if the PEs reside on a shared LAN. On the other hand, saving inter-PE messages at the cost of some extra overhead at the PEs is worthwhile in a WAN environment. Therefore, simulation techniques should be chosen based on prevailing variable values in the simulation.

Any switching of technique requires a particular switching point which is agreed upon by all entities. We can call this as the "clean-slate" time which means that this time is such that PEs contain within themselves all information about their states and can proceed smoothly should there be a change in simulation technique.

The super-step boundary in our techniques provides the PEs with such "clean-slate" points. At each super-step boundary the participating PEs calculate the next super-step size and inform the host about their predicted values. The Host then uses the mean of these predicted values as the super-step sizes for time-stepped execution. We abstract the number of future events by a probabilistic approach. Since the events occurring in the non-overlapping super-steps are assumed to be independent, the Poisson probability density function is used in our prediction. The chi-square test with a 95% confidence measure is used to ensure the goodness-of-fit. In cases where a Poison fit is not available we resort to a linear average for estimating the next super-step size.

5.1 Mathematical Estimate of the Super-Step

We estimate the next super-step size (ℓ^{n+1}) based on the observed pattern of events so far. The analysis assumes that the simulation has advanced *t* time units. Let these *t* time units comprise say *n* number of simulation super-steps each of variable size.

If, from previous knowledge, we estimate that an event count of 'χ' per simulation super-step is acceptable then we can calculate ℓ^{n+1} using:

$$\ell^{n+1} = \left\{ \begin{array}{l} \dfrac{\chi}{2*\kappa+\lambda} \quad \text{(when linear fit is used)} \\ \dfrac{\chi}{2*(1-e^{-\lambda})+\dfrac{S|_{\Psi}}{|\Psi|}+\lambda} \quad \text{(when Poisson fit is used)} \end{array} \right\}$$

where Ψ denotes the history period being observed (with $|\Psi|$ denoting the length of Ψ), κ is the observed event coverage, $S|_{\Psi}$ is the number of synchronization points in Ψ and λ is the observed event density in Ψ (Please refer to Appendix: Super-step size estimation for details).

6 EXPERIMENTS AND RESULTS

When translating the conceptual model into a computer program, the modeler adopts a particular world-view or orientation. The world-view may be dictated by the simulation language or package chosen for the implementation or constitutes a design choice when implementing in a general-purpose language. The most common world-views for DES are: event oriented modeling, process oriented modeling, and activity scanning modeling (Banks, Carson, and Nelson 1996). The application domain used in this paper is war game simulation where combating federates interact with each other, and the activity scanning world-view was adopted.

6.1 The Simulation Testbed

The activity scanning conceptual framework (Buxton and Laski 1962) was chosen for implementing the simulation due to its common use in simulations that use fixed time increments (time-steps). The flow mechanism (Page 1997) adopted by such simulations is shown in Figure 9. One evaluation of each condition represents a single scan. If any condition is satisfied in a scan then the complete set of conditions must be re-scanned because the actions performed due to the previous condition being satisfied might have caused some new conditions to be satisfied. If no conditions are satisfied in a single scan then Phase 2 terminates.

Our technique aims to improve the efficiency of simulations by reducing the synchronization intervals when the event coverage is not dense. Parameters that need to be considered are the savings introduced by the new technique in the number of synchronizations, number of overhead messages exchanged and the condition, namely the event coverage, under which these savings (if any) were observed. These parameters have been chosen based on current literature (Fujimoto 1989 and Nicol 1993).

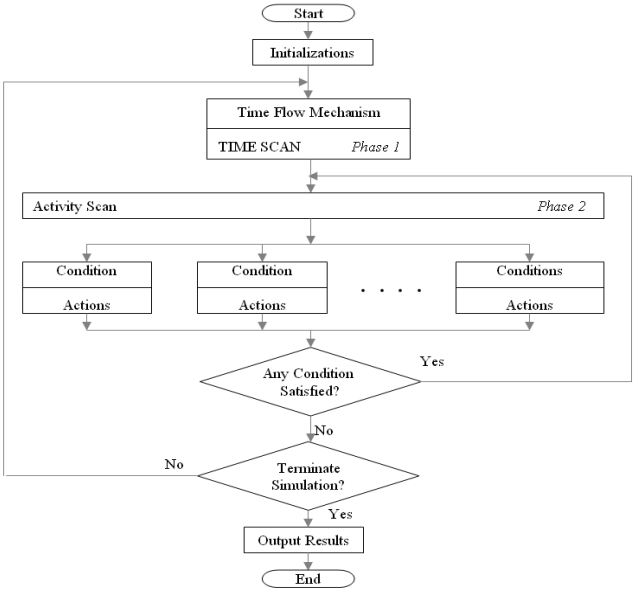

Figure 9: Activity Scan Flow Diagram

The traditional time-stepped mechanism was used as the benchmark. Figure 1 shows the number of synchronization points and overhead / time advance messages involved in the conventional technique. It is quite evident from the figure that two overhead events are required per PE per unit simulation time for proper synchronization when the traditional technique is used. Since synchronization occurs at every time step the number of synchronizations are equal to the number of time steps.

The activity scanning technique usually employs entities which are made up of several models. Each model is assigned a current state which is changed when an event targeted to that particular model is received either from another model within itself or from a foreign model residing in another entity. Our experimental evaluation was done using four-, eight- and sixteen-PE setups. Each participating entity in the simulation contains the following three distinct types of models:

- Action models (models that encapsulate motion)
- Behavioral models (viz. models that simulate patrolling, scanning for target etc)
- Physical models (models that actually change physical characteristics of an entity such as location, color, capabilities etc).

For our experiments we used entities built of 5 behavioral models, 1 physical and 1 action model. These

models when combined with their ability to generate events at random gave us sufficient flexibility to vary the Event Coverage.

6.2 Results and Analysis

The optimized technique was evaluated with respect to the number of overhead messages (as compared to the traditional time-stepped technique) and the physical time overhead at the host due to the introduction of the FTL.

6.2.1 Overhead Messages

Figures 10, 11 and 12 show the average cumulative super-step boundaries, overhead messages and time advances respectively across 8 PEs participating in the simulation over time for different values of event count and event coverage.

The graphs display two important properties, *viz.* response to changes in the super-step throttle and the response to variation in the event coverage.

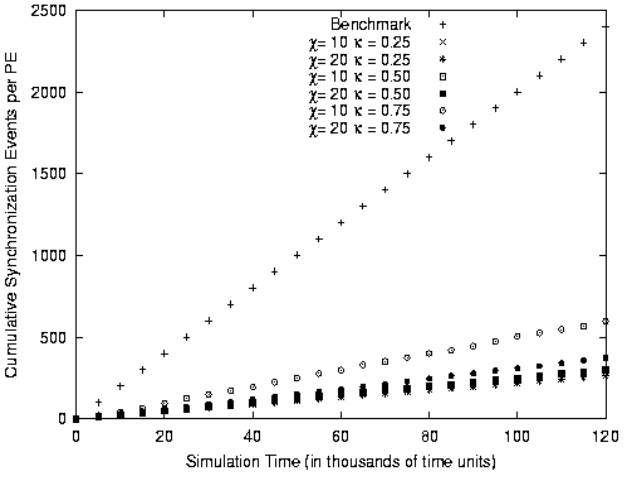

Figure 10: Cumulative Super-Step Boundaries (8 PEs)

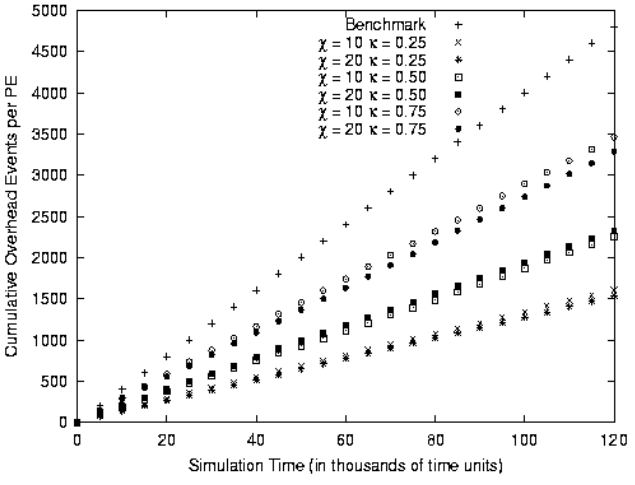

Figure 11: Cumulative Overhead Events (8 PEs)

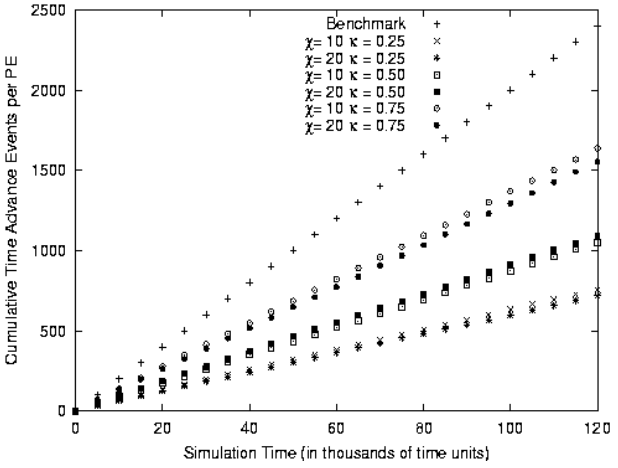

Figure 12: Cumulative Time Advances (8 PEs)

6.2.1.1 Response to Super-Step Throttle

The super-step throttle, as derived in the analysis section, dictates the size of the super-step used by the technique. On the positive side, large values can lead to an increase in size of the super-step and thus potentially delay synchronizations that involve all the participating PEs, as can be observed from Figure 10.

However, large super-steps preclude that the event coverage will not change within the span of the super-step and leads to wider periods in which the simulation technique cannot be changed.

An inordinately large sized super-step is potentially dangerous because event density that is randomly varying may exponentially rise during the course of the super-step and the PEs cannot vote for a change in the simulation technique until the completion of the current super-step. Consider the following example: At time T=200 if we decide to use a super-step of size 18 and at T=204 we have a sudden burst in the event density. It is now not possible to switch to the traditional time-stepped mode of simulation until T= 218.

To overcome this potential hazard we introduce a threshold value for the maximum allowable number of time steps that can elapse between two mandatory global synchronizations. This threshold, denoted by the variable MAX_SUPER_STEP, was set to 10 throughout our experiments.

6.2.1.2 Response to Event Coverage

From the graph for overhead events (Figure 11) it is clear that the number of synchronizations and the number of overhead messages are less than the benchmark in all cases. This can also be seen from the analysis shown in the previous section. Recall that the worst-case overhead based on linear regression from the previous section was shown to be 2 events per time-unit versus 2 events per time unit of

the benchmark algorithm. Hence, when the event coverage equals 1, the performance of our proposed technique equals that of the conventional technique. This is evident from the experimental values obtained. The number of overhead messages, which is a crucial consideration factor in distributed simulations running on a LAN/WAN, is proportional to the event coverage in the proposed technique.

Experiments were repeated using four nodes and sixteen nodes and the observed decrease in case of four participating PEs (or increase in case of the sixteen PEs) in the number of overhead messages was proportional to the decrease (or increase) in the number of PEs participating in the simulation.

6.3 Overhead Time

The introduction of the FTL results in some overhead at the host. Since we are reducing the number of overhead messages at the cost of introducing an additional data structure at the central host it is important to measure the physical time spent by the host in maintaining the data structure. This time is affected by two parameters –the number of time stamps already in the FTL and the number of time stamps newly introduced. We varied these two parameters from 0 to 500 and observed (Figure 13) that when both the parameters were varied beyond 350, the physical time taken ranged between 0.005 to 0.020 seconds.

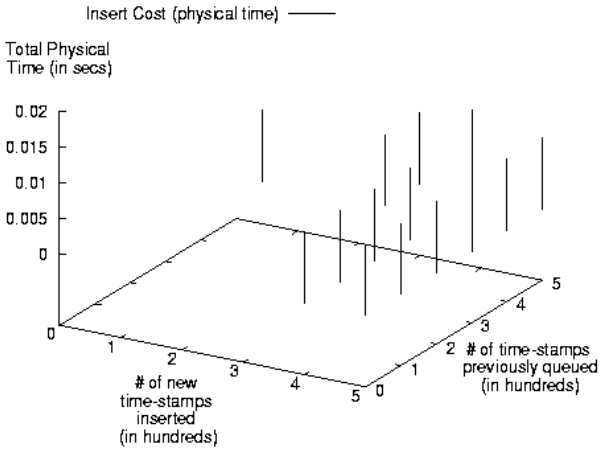

Figure 13: Overhead Physical Time at Central Host

The overhead time consumed for 'piggybacking' information on to the *Ready* messages was also measured. However, the time consumed was negligible (<10 msec) even when 1000 new time-stamps were piggybacked. Figure 14 shows the observed results.

7 FURTHER WORK AND CONCLUSIONS

This paper introduces an optimization to traditional time-stepped simulations, particularly in the domain of war

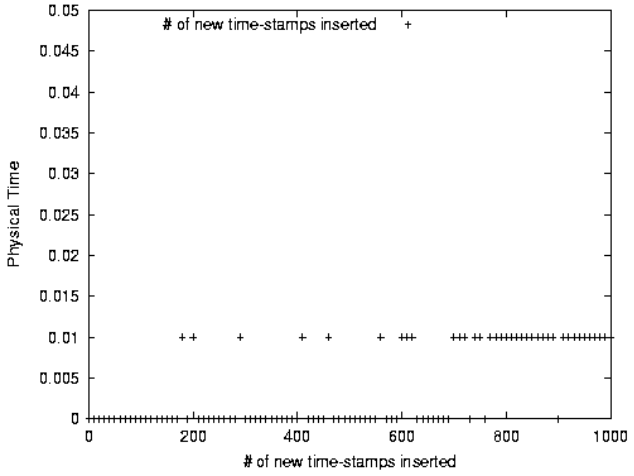

Figure 14: Overhead Physical Time at PEs

game simulations. A super-stepping mechanism is also introduced so as to facilitate swtching between simulation techniques to ensure efficiency. Our experimental results show that substantial savings in overhead can be achieved when this piggy-backing optimization is used in simulations when compared to the conventional time-step technique. The proposed mechanism is simple and robust.

Though the optimization suggested in this paper addresses simulations with sparse event coverage it does not provide for utilization of any available lookaheads between participating entities. Our further work will focus on building a probabilistic super-step based model to efficiently simulate entities with sparse event sets and with intermittent availability of lookaheads while fundamentally maintaining the time-stepped paradigm.

ACKNOWLEDGMENTS

This research was supported by the NUS-Ministry of Defence, Singapore Collaboration project GR6757.

APPENDIX: SUPER-STEP SIZE ESTIMATION

Let the average simulation event density of a PE be represented by ∂. Thus, the event densities of PE_1, PE_2 … PE_n will be represented by ∂_1, ∂_2 … ∂_n respectively. Further, let the superscript denote the simulation time unit to which the event density corresponds. For instance the event density of PE_2 in simulation time unit 't' will be denoted as ∂_2^t. Since each super-step consists of one or more simulation time units it is important to represent the length of a super-step by a variable. Let ℓ^n represent the length of the n^{th} simulation time super-step in units of simulation time. Finally, since most of our predictions about the future will be based on the observed behavior from the past we denote

the length of the history period considered (in simulation time units) as $|\Psi|$ and the history period itself as Ψ.

For each PE we have two types of events – overhead events and simulation events represented by $\varsigma_{n+1}^{overhead}$ and ς_{n+1}^{sim} respectively. Overhead events are those events that are used to support the simulation. Synchronization costs, fault-tolerance costs are some examples of overheads. Simulation events are those that actually form a part of the simulation and modify the system state in some way or another.

Estimating $\varsigma_{n+1}^{overhead}\big|_{\Psi}$ (overhead events in Ψ):

We abstract the following two overheads:

- Cost incurred by each PE during the history period in informing the centralized host about the time-stamps of future events generated by it, represented by $\varsigma_{n+1}^{send}\big|_{\Psi}$
- Cost incurred in informing the PEs to advance in simulation time, represented by $\varsigma_{n+1}^{receive}\big|_{\Psi}$.

The total overhead cost incurred across Ψ is the sum of $\varsigma_{n+1}^{send}\big|_{\Psi}$ and $\varsigma_{n+1}^{receive}\big|_{\Psi}$. Therefore,

$$\varsigma_{n+1}^{overhead}\big|_{\Psi} = \varsigma_{n+1}^{send}\big|_{\Psi} + \varsigma_{n+1}^{receive}\big|_{\Psi} \qquad \text{(A-1)}$$

The ready for time advance events sent to the host from the entities are replied with one single safe time declaration event by the host. In addition, there might not be any event at a PE at the boundary of a super-step). In the worst case, all boundaries of super-steps in the history period may not have an event to be executed at an PE. Therefore,

$$\varsigma_{n+1}^{receive}\big|_{\Psi} \le \varsigma_{n+1}^{send}\big|_{\Psi} + S\big|_{\Psi} \qquad \text{(A-2)}$$

where $S\big|_{\Psi}$ is the count of synchronization points in Ψ.

Using (A-1) and (A-2) we have:

$$\varsigma_{n+1}^{overhead}\big|_{\Psi} \le (2 * \varsigma_{n+1}^{send}\big|_{\Psi}) + S\big|_{\Psi}. \qquad \text{(A-3)}$$

The χ^2 test either supports the hypothesis that the observed events in the history period obeys a Poisson distribution or rejects it.

- Case 1: Successful Poisson fit

  Since we have a successful Poisson fit we use the Poisson mean represented by λ.

  $\varsigma_{n+1}^{send}\big|_{\Psi}$ = Ready events sent to Host

  = P [Time-step has events]*$|\Psi|$

  = (1 – P [$\partial_n = 0$]) *$|\Psi|$

  = $(1-e^{-\lambda})*|\Psi|$.

  For ease of analysis we reduce the result in (A-3) to an equality by considering only the worst case behavior. Hence,

  $$\varsigma_{n+1}^{overhead}\big|_{\Psi} = 2*(1-e^{-\lambda})*|\Psi| + S\big|_{\Psi}. \qquad \text{(A-4)}$$

- Case 2: Unsuccessful Poisson fit

  For this case, we use a simple linear estimate. This rudimentary method does not provide us any means of knowing what the probability that a particular entity has an event to be executed in any simulation time unit is. Hence we estimate the overhead directly based on the event coverage 'κ' observed during the history period. Therefore,

  $$\varsigma_{n+1}^{overhead}\big|_{\Psi} = 2 * \kappa * |\Psi|. \qquad \text{(A-5)}$$

  This is because every time stamp that has an event involves a synchronization message and a ready message exchanged between the entity and the host.

Estimating $\varsigma_{n+1}^{sim}\big|_{\Psi}$ (simulation events in Ψ):

We base our estimation, like before, on the observed events in the history period. Firstly, the total simulation event count in the history period is :

$$\varsigma_{n+1}^{sim}\big|_{\Psi} = \sum_{t\in\Psi} \partial_n^t \ . \qquad \text{(A-6)}$$

This is the same as the product of mean observed event density during the history period and the length of the history period. That is:

$$\varsigma_{n+1}^{sim}\big|_{\Psi} = \lambda * |\Psi|. \qquad \text{(A-7)}$$

Using the above results we can now calculate the length of the next super-step that will yield us the desired number of total events in a super-step. For instance, by previous observations if we have concluded that the nearness to real world offered by an event count of 'χ' per super-step is acceptable then we can estimate ℓ^{n+1} using:

$$\ell^{n+1} = \frac{\chi * |\Psi|}{(\varsigma_{n+1}^{overhead}\big|_{\Psi} + \varsigma_{n+1}^{sim}\big|_{\Psi})} \qquad \text{(A-8)}$$

where 'χ' is the desired number of total events in a super-step. Equation (A-8) follows from the fact that $\varsigma_{n+1}^{overhead} + \varsigma_{n+1}^{sim}$ number of events have been observed in number of time-steps and we would like to see number of events in the next super-step of length ℓ^{n+1}.

Substituting (A-4)/(A-5) and (A-7) in (A-8) and simplifying, we have:

$$\ell^{n+1} = \left\{ \begin{array}{l} \dfrac{\chi}{2*\kappa+\lambda} \quad \text{(when linear fit is used)} \\ \\ \dfrac{\chi}{2*(1-e^{-\lambda})+\dfrac{S\,|_{\Psi}}{|\Psi|}+\lambda} \quad \text{(when Poisson fit is used)} \end{array} \right\}. \quad \text{(A-9)}$$

REFERENCES

Ayani, R. 1989. A parallel simulation scheme based on distance between objects, *Proceedings of the SCS Multiconference on distributed simulations*. Volume 21, Pages 113-118.

Banks, J., Carson II, J.S., and Nelson, B.L. 1996. Discrete-Event System Simulation, Second Edition, Prentice Hall.

Buxton, J. N., and Laski J. G. 1962. Control and Simulation Language, *Computer Journal*, Volume 5(3), 1962.

Cai, W., Letertre, E., and Turner, S.J. 1997. Dag Consistent Parallel Simulation: a Predictable and Robust Conservative Algorithm, *Proceedings of the 11th Workshop on PADS*, pp. 178-181.

Chandy, K.M., and Misra, J. 1981. Asynchronous Distributed Simulation via a Sequence of Parallel computations. *Communications of the ACM*, Volume 24(11), pp. 198-206.

Dickens, P.M., and Reynolds Jr., P.F. 1991. A Performance Model for Parallel Simulation, *Proceedings of the Winter Simulation Conference*, pp. 618-626.

Eick S.G., Greenberg A.G. Greenberg, Lubachevsky B.D. and Weiss, A. 1993. Synchronous Relaxation for Parallel Simulations with Applications to Circuit-Switched Networks, *ACM Transactions on Modeling and Computer Simulation*, Volume 3(4), pp. 287-314.

Fujimoto, R.M. 1989. Performance Measurements of Distributed Simulation Strategies, *Transactions of the Society for Computer Simulation*, Volume 6, Pages 89-132.

Gan, B.P., Low, Y.H., Cai, W., Turner, S.J., Jain, S., Hsu, W.J., and Huang, S.Y. 2001. The Development of Conservative Superstep Protocols for Shared Memory Multiprocessor Systems, *Parallel and Distributed Computing Practices*, Volume 4(1), pp. 1-17.

Groselj B., and Tropper C. 1988. The Time-of-Next-Event Algorithm, *Proceedings of the SCS Multiconference on Distributed Simulation*, pp. 25-29, Society for Computer Simulation.

Nicol, D.M. 1993. The cost of conservative synchronization in parallel discrete event simulations, *Journal of the ACM*, Volume 40(2), pp. 304-333.

Lubachevsky, B.D. 1988. Bounded Lag Distributed Discrete Event Simulation, *Proceedings of the SCS Multiconference on Distributed Simulation*, Pages 183-191, Society for Computer Simulation.

Lubachevsky, B.D. 1989. Scalability of the bounded lag distributed discrete event simulation, *In Distributed Simulation*, Volume 21(2), pp. 100-107.

Page, E.H. 1997. Zero Lookahead in a Distributed Time-Stepped Simulation, *Simulation Digest*, Volume 26(2), pp. 4-13.

AUTHOR BIOGRAPHIES

SENG CHUAN TAY graduated from the National University of Singapore (NUS) with a Ph.D. degree in 1999. His research interests include parallel and distributed simulation (mechanism and modeling) and satellite image applications. He is also the Deputy Director of The Centre for Remote Imaging, Sensing and Processing (CRISP) at NUS. He can be reached at <crstaysc@nus.edu.sg>.

GARY S.H. TAN is a Senior Lecturer at the School of Computing, National University of Singapore (NUS). His research interests are Parallel and Distributed Systems, Parallel and Distributed Simulation, and High Level Architecture. He is a member of the Modelling and Simulation Group at NUS and a member of the ACM and IEEE Computer Society. His email address is <gtan@comp.nus.edu.sg>.

KARTHIK SHENOY is a Masters student at the School of Computing, National University of Singapore. He received his B.E. degree in Computer Science from Mangalore University, India in 1999. His research interests are in the domain of parallel and distributed systems. His email address is <karthiks@comp.nus.edu.sg>.