Proceedings of the 1996 Winter Simulation Conference
ed. J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain

LOGICAL PROCESS SIZE IN PARALLEL SIMULATIONS

Fang Hao
Karen Wilson
Richard Fujimoto
Ellen Zegura

College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280

ABSTRACT

Most existing synchronization protocols require that the simulation application be partitioned and mapped to logical processes to make it suitable for parallel execution. Assuming the simulation models some number of physical components, an important design question is how many components should be mapped to each logical process? This is a non-trivial question because logical process "size" affects the efficiency of the synchronization protocol, load balance, and approach for implementing shared state variables, as well as the efficiency of the event processing loop within the parallel simulator. This question is studied in the context of a Time Warp-based parallel simulator. Results of two experimental studies are described that compare the performance of parallel simulators using different logical process sizes. One study uses synthetic workloads; the other uses an Asynchronous Transfer Mode (ATM) network. These results show that the choice of logical process size can have a significant effect on performance, and the optimal size depends on model size, the number of available processors, and detailed semantics of the model itself.

1 INTRODUCTION

It is well known that long execution times severely limit the effectiveness of discrete event simulation techniques in modeling moderate and large sized systems. One approach to reducing execution time is to use parallel discrete event simulation (PDES) techniques to execute the simulation model (Fujimoto 1990b, and Nicol and Fujimoto, 1994). This is a particularly attractive approach when simulating large networks because such networks inherently contain large amounts of spatial concurrency. Here, we are primarily concerned with *optimistic* synchronization algorithms such as Time Warp (Jefferson 1985) that utilize a rollback and recovery procedure to synchronize the computation.

Traditional PDES approaches such as Time Warp define the simulation as a collection of logical processes (LPs) that execute concurrently, and communicate by exchanging timestamped *events*, or *messages*; these two terms are used synonymously here. For example, in an ATM network simulation, logical processes must be defined to model components in the network, e.g., traffic sources, switches, and multiplexers. Thus an important design question is: How should the simulation model be partitioned into logical processes? Is it better to define many small logical processes, e.g., a single switch might be modeled as several logical processes, or should one define "large" logical processes that include several components (e.g., several switches)? As elaborated upon later, this is a non-trivial question because of considerations such as load balancing, shared state, and interactions between the synchronization algorithm and the way the network has been partitioned. We refer to this question as the *logical process size* problem.

Here, we are especially concerned with *small-granularity* simulations containing only a modest amount of computation within each event, e.g., a few hundred machine instructions. Simulations of telecommunication networks are often small-granularity simulations because typical events might do little more than update queue length state variables and schedule new events. It is particularly important that parallel simulator overheads be minimized for these simulations because only a modest amount of overhead computation for each event could significantly degrade performance.

After considering a synthetic workload, we focus on simulation of ATM networks. Modern high-bandwidth networks must provide acceptable quality of service to an increasingly diverse set of applications and traffic sources ranging from low speed

data transfers to high quality high-definition television (HDTV) distribution. ATM is one method for accommodating diverse applications on an integrated network. This diversity among sources has a direct impact on the design of high-speed networks and their expected performance. Traditional traffic assumptions are not applicable at the ATM *cell* (i.e., 53 byte fixed size packet) level (Das et al. 1994). Thus, the analytical approach to performance modeling is generally numerically intensive and often approximate. Further, the approximate analytical models require validation that is typically performed by simulations. In any case, simulation is an indispensable tool when it comes to testing the performance of a system over a wide variety of traffic loads and scenarios. However, simulations of high-speed networks require excessively long execution times, since network designers and researchers require simulations of large networks, and often have to collect statistics over minutes and hours of real time operation.

In the next section we review related work in accelerating the execution of simulations of telecommunication networks. Issues and tradeoffs concerning the partitioning of network simulations are then discussed and evaluated experimentally. A case study is described comparing the performance of *two* parallel simulations of the same ATM network model, one using "large" LPs, and the second using "small" ones. The performance of these two simulations of the same network model on a common parallel simulation platform are then compared experimentally.

2 RELATED WORK

A considerable amount of work in the circuit design community has been concerned with defining *partitioning algorithms* that divide circuits into subcircuits for simulation on parallel computers, e.g., see Davoren (1989), and Nandy and Loucks (1992). There, the partitioning problem is typically posed as decomposing a large circuit into N sub-circuits, where each sub-circuit is simulated on a different processor. These algorithms usually assume it is best to operate with only one logical process per processor, and do not directly address the question of how logical process size affects parallel simulation performance.

A goal of the work described here is development of a flexible, widely-applicable, parallel simulation tool for modeling telecommunication networks. This goal has been pursued by other researchers as well. For example, Earnshaw and Hind (1992) report up to an order of magnitude speedup in simulating B-ISDN networks. Gaujal et al. (1993) report an order of magnitude speedup in simulating call routing using application specific parallel simulation techniques, and report simulator performance exceeding 3,000,000 simulated calls per minute. Turner and Xu (1992) report success in speeding up telephone switching network simulations using a variation of Time Warp. More recently, work at the University of Calgary is developing a parallel simulation toolkit for modeling telecommunication networks (Unger and Xiao 1994). Not all report success, however. Phillips and Cuthbert (1991) report that their Transputer-based conservative simulations ran more slowly than a sequential simulator. Tallieu and Verboven (1991) also report disappointing performance in simulating an Ethernet. The central contribution of our results is in addressing the LP size question in the context of parallel simulators such as those described above. Although the experimental results presented in this paper apply exclusively to Time Warp simulations, some of the qualitative tradeoffs discussed in this work also apply to conservative simulation algorithms.

3 LOGICAL PROCESS SIZE

The physical system is assumed to be composed of a collection of *components*, where each component is informally defined as the most primitive element whose behavior is explicitly represented in the simulation model via state variables and events. Perhaps a more pragmatic definition of a component is the smallest portion of the simulation model that could be represented by a logical process, given the level of detail required to meet the goals of the simulation study. The components of the cell-level ATM simulation model described later include traffic sources, traffic sinks, and elements of each switch (e.g., output ports).

The central question to be addressed here concerns the number of components that should be included in a single logical process to maximize performance. The advantages of defining "large" logical processes that contain many components include:

- *efficient shared state.* State variables accessed by simulators for different components within the *same* logical process may be directly accessed by those components using local memory references.
- *event combining.* If it is known at model development time that an event will occur at several different components at the same instant in time, and these components are all modeled within a single LP, then the simulator need only schedule a *single* event at the LP.
- *process-oriented simulations.* Process-oriented simulators usually define each LP to contain a

single thread of execution that can "pause", e.g., to wait for another event. This is in contrast to event-oriented simulators where each event results in a procedure to be executed, and simulated time only advances between procedure calls. Implementation of process-oriented simulations typically requires a runtime stack to be maintained for each LP, and light-weight threads to switch execution among the LPs. Simulations using a small number of large LPs will typically require much less memory than simulations using a large number of small LPs. Due to reduced memory management overhead (e.g., caches operate more efficiently for small programs), large LP simulations may yield better performance as well as lower memory consumption.

The efficiency of the event scheduling mechanism may be affected by LP size, depending on the implementation of the parallel simulation executive. Event scheduling involves (1) locating the pending event list for the LP receiving the message, and (2) enqueueing the message into this data structure. Interprocessor communication will also be required if the sending and receiving LPs are mapped to different processors, however, this affects large and small LP simulators equally, assuming the same mapping of components to processors is used. If (1) above is faster when an LP schedules an event for itself compared to scheduling an event for a different LP, using large LPs offer some benefit because there will typically be more of these "self-scheduled" events. However, it is not a priori clear that self-scheduled events are necessarily faster than scheduling events between different LPs on the same processor. We conjecture that for most parallel simulation executives, large LPs offer little or no advantage over small LP simulators with respect to event scheduling because locating a destination LP should not be a time consuming operation in a well designed simulation executive. For example, in Das et al. (1994) locating the destination LP is a constant time operation (essentially, indexing into an array), independent of the destination or the number of LPs. Further, because a large LP including several components will typically have more pending events than a small LP, one would expect longer enqueue times, (2) above, giving the edge to small LP simulators.

In addition to scheduling events, each processor must repeatedly select the next event to be processed, typically by selecting the event with the smallest timestamp among those LPs mapped to the processor. We refer to the computation to perform this task as the *event selection overhead.* Event selection typically involves (1) selecting an LP to execute next, and (2) selecting the smallest timestamped event within that LP. Use of large LPs will typically reduce (1) because there are fewer LPs to manage, but at the cost of increasing (2), because there will usually be more pending events within each LP. If data structures with similar access times (e.g., $O(logN)$ insertion and deletion time priority queues) are used for maintaining the list of runnable LPs and the list of pending events within each LP, LP size may not significantly affect event selection overhead. Similarly, if both lists are combined into a single priority queue data structure as described in Das et al. (1994), LP size will not affect the event selection overhead at all.

"Small" logical processes do offer some significant advantages over large LPs:

- *greater potential concurrency.* Concurrency is limited by the number of LPs, so smaller LP simulators may offer more potential concurrency.
- *more flexible load distribution.* Better workload balance may be obtained because the granularity of computation that may be moved from one processor to another is finer.
- *potentially lower state saving overhead.* Optimistic protocols require snapshots of the LP's state to be made to enable rollback. Large LPs will typically contain more state than smaller ones, necessitating more state copying.
- *less rolled back computation or artificial blocking.* If two different, independent components are combined into a single larger LP in a Time Warp simulation, a rollback in one component may unnecessarily cause rollback events in the second component. Similarly, in a conservative simulation, blocking one component may unnecessarily delay processing events in another component within the same LP.

From the above discussion, we see that it is not a priori clear whether large or small LPs will yield the best performance, or how much difference this question makes with respect to overall performance. It is clear that the answer to this question depends on the simulation model and implementation details of the underlying simulation executive.

4 SYNTHETIC WORKLOAD

To quantitatively evaluate how different factors affect performance, a synthetic workload program called PHOLD was used (Fujimoto 1990a). The simulation model consists of a collection of entities, that send messages to each other. Each entity sends message to

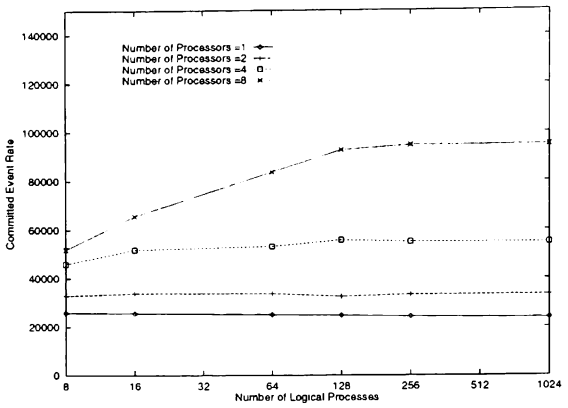

Figure 1: Events committed per second–PHOLD

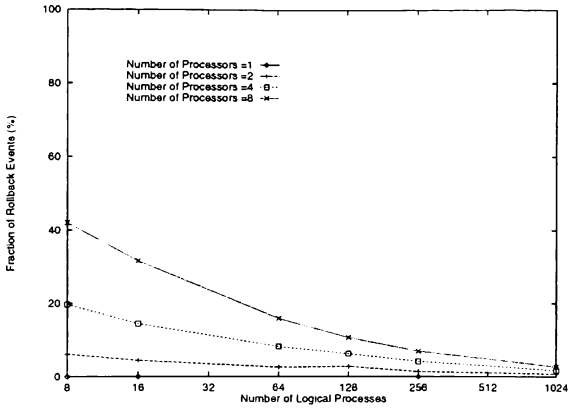

Figure 2: Rollbacks–PHOLD

another entity after processing each event. The destination is selected from a uniform distribution. The time stamp increment is selected from an exponential distribution. The entities are partitioned into logical processes so that each LP contains the same number of entities. The workload of the PHOLD simulator is specified as the message population, i.e., the number of unprocessed events that exist in a sequential execution of the simulator at any instant.

To emulate the effect of shared state, state update messages are added to the PHOLD model. The entities are clustered into a number of groups. All entities in the same group share state variables. The shared state is updated at simulated times following a Poisson distribution. In each group, one "master" entity is chosen that is responsible for sending update messages to the other entities in the group. If all entities in a group are in the same logical process, only one update message needs to be sent to implement each update, i.e., the master entity only sends one update message to itself. However, if the entities in a group span several logical processes, multiple update messages must be sent, i.e., the master entity must not only send an update message to itself, but also send one update message to each LP containing entities within the group.

4.1 Experiments without State Updates

The first set of experiments were conducted to evaluate the effect of rollbacks on the performance of PHOLD without state updates, under different LP sizes. The performance of PHOLD, as measured by the number of events committed per second with different numbers of logical processes (from 8 to 1024) and processors (1, 2, 4, or 8), are shown in Figure 1, where the message population is 64.

We observe that the message event rate increases with the number of logical processes, i.e., the fine-grained simulator produces better performance than the coarse-grained simulator. The principal reason is the coarse-grained simulator is prone to having more unnecessary rollbacks than the fine-grained simulator. We also observe that this effect is more pronounced when the number of processors is increased. This is because increasing the number of processors results in more optimistic execution, accentuating the rollback behavior.

Figure 2 shows the fraction of processed events that are rolled back. This graph confirms that the number of rollbacks increases significantly as the LP size increases (fewer LPs).

However, when the workload is heavy (e.g., message population is 1024), we observe that the number of logical processes does not have a dominant effect on the message event rate. This is because the heavy work load reduces the rate of advance of each LP and thus reduces the number of rollbacks.

4.2 Experiments with State Updates

A second set of experiments were conducted to evaluate the effect of state updates on performance. The performance was measured by determining the number of "effective" committed message events per second, where the number of "effective" events is defined as the number of total committed events minus the number of update events. The latter term represents an additional overhead incurred by simulations using small LPs. Figure 3 shows performance with different numbers of logical processes(from 8 to 1024), processors(4 or 8), and mean state update intervals (equal to 1 or 10 times the timestamp increment, or no update messages). The message population is 64.

We observe that: (1) In general, the effective event rate increases, and then declines, as the number of

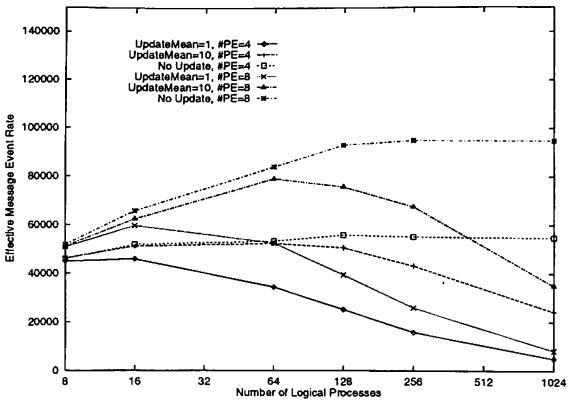

Figure 3: Effective events per second–PHOLD

logical processes is increased. This is because when the number of LPs is small, the number of update messages is small, so rollback is a dominant factor affecting performance; however, when the number of LPs is large, the number of update messages is large, so the overhead for update messages becomes dominant. (2) Performance is more sensitive to changes in the number of LPs when the mean value of the update interval is smaller, because more update messages are generated.

Again, when the workload is heavy (e.g., message population is 1024), performance is less sensitive to changes in the number of LPs, similar to what was observed in the previous experiments.

5 CASE STUDY: ATM SIMULATION

The simulation of large networks of ATM switches presents an interesting case study for optimal selection of logical process size. A single ATM switch is commonly abstracted as a set of output queues, one per output link. A large network of switches can then be partitioned into logical processes spanning a variety of process sizes, including (1) one process per output queue, (2) one process per switch, and (3) multiple switches per process. For this case study, we focus on the first and third approaches; our *fine-grained* (small LP) implementation uses one process per output queue while our *coarse-grained* (large LP) implementation uses multiple switches (and hosts) per process, where each host represents a single source of network traffic, and a single sink.

In order to compare the results from these two partitioning approaches, we maintain the following characteristics across the two implementations: (1) A switch is represented by a set of output queues, one per switch output link. (2) We use a common topology for the large network of switches, which is taken from an existing IP network called *Sesquinet*. The network consists of 94 switches and 80 hosts. Each host is represented by a source and a sink. (3) Traffic is generated using a uniform geometric bursty traffic model. The destination of each burst of cells is selected from a uniform distribution. All sources behave in a similar fashion. (4) The routing is shortest path, computed using unit weights on all links. (5) The simulators only model forwarding of ATM cells from one component to another. A static set of circuits is assumed, i.e., no new circuits are constructed during the simulation, and circuits are never torn down. (6) To emulate the effect of shared state (e.g., due to creating or destroying circuits during the simulation that cause routing table entries to change), both simulations use a common state update model. Specifically, state updates are generated using a Poisson process, and affect the state of a single switch. (7) Both simulators use identical mappings of logical processes to processors in the sense that the set of components modeled by LPs mapped to a single processor is the same for both simulators. The fine-grained simulation used here does not take advantage of the fact that the LPs used to model a single switch could be distributed over multiple processors.

We now describe the details particular to each of the two implementations used to study process granularity.

5.1 Fine-Grained Implementation

The fine-grained implementation uses a large number of small LPs, associating each host and each output queue in a switch with a separate LP. The fine-grained implementation of the Sesquinet has 436 LPs.

The fine-grained simulation uses copy state saving exclusively. Each state update event (pertaining to a single switch) results in the generation of additional state update events that are sent to other LPs modeling different output ports of the switch; the events are required to ensure the different copies remain consistent. This strategy for updating state is sometimes called "push" processing, in contrast to "pull" processing where LPs do not maintain local copies, but rather, state is explicitly requested by each LP via "query events" when needed (Wieland and Jefferson 1989). Push processing is more efficient when state updates occur less frequently than state queries.

5.2 Coarse-Grained Implementation

The coarse-grained implementation creates a small number of large LPs, grouping multiple sources together into source LPs, multiple sinks together into sink LPs and multiple switches together into switch

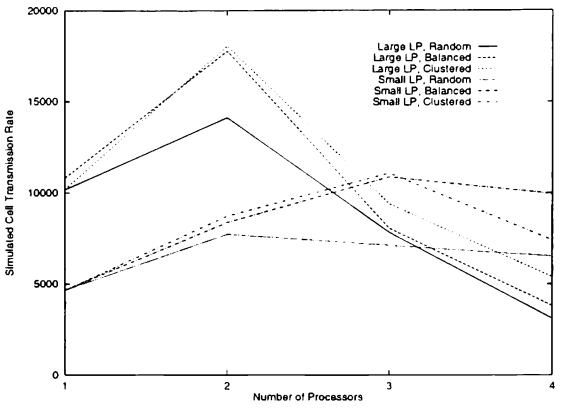

Figure 4: Simulated cell transmissions–ATM

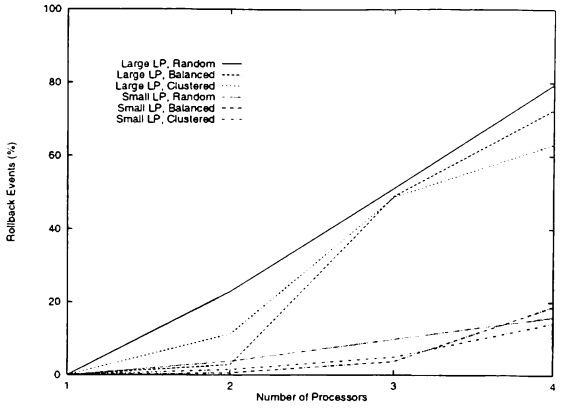

Figure 5: Rollbacks–ATM

LPs. In the coarse-grained implementation of the Sesquinet there is one source, one sink, and one switch LP per processor.

The implementation of the source model results in an event at each source during each time unit that determines whether the source turns on or off. The synchronous nature of this model results in many events across the different sources occuring at the same simulated time. The event handling routine in the coarse-grained simulator uses event combining (discussed earlier) and only schedules one event for all of the sources modeled by the LP. By contrast, the fine-grained simulator described earlier must schedule separate events for each source LP.

The coarse-grained simulator uses copy state saving for variables that are modified with each event and incremental state saving for other state variables. Because incremental state saving is somewhat more expensive than copy state saving, this situation favors fine-grained LPs. State updates are handled more efficiently in large-grained LPs, since all of the modified state is confined to a single LP.

6 PERFORMANCE RESULTS

The two parallel simulators were executed on a four processor shared-memory Sparc multiprocessor. Each processor is a 50 MHz HyperSparc. The performance metric used in this study is the number of simulated cell transmissions (sending a cell over a single communication link) per second. The Time Warp executive used in this study is described in Das et al. (1994). The first set of experiments does not consider state updates. The effect of the frequency of state updates on performance is then studied in a second set of experiments.

6.1 Experiments without State Updates

A sequence of experiments were performed using the following partitioning algorithms: (1) *Random mapping:* each switch or host node in the network is randomly mapped onto a processor. (2) *Balanced random mapping:* each node is randomly mapped onto a processor, with the constraint that each processor must be assigned an equal, total number of switch and host nodes. (3) *Clustering mapping:* the network is partitioned to balance the workload (as measured in the number of hosts and switches mapped to each processor) and to reduce interprocessor communications. The partitioning was performed manually.

The performance of the two simulators under different partitioning strategies with light traffic load is shown in Figure 4. The probability that a source is in the "on" (transmitting) state is 0.06; similar results were observed when the traffic load is high (e.g., the "on" probability is 0.67). Across the experiments, we observe that (1) The coarse-grained simulator far outperforms the fine-grained simulator for one or two processors. (2) The fine-grained simulator's performance approaches, and in some cases surpasses the coarse-grained simulator for three or four processors. (3) Among the three mapping strategies, clustering mapping yields the best performance, and random mapping the worst performance. Moreover, the partitioning algorithm does have a significant affect on performance in both simulators. (4) All of the simulators exhibit some degree of performance degradation as the number of processors increases, as elaborated upon below.

The principal reason the coarse-grained simulator outperforms the fine-grained simulator when using a small number of processors is because it can exploit event combining. This significantly reduces the amount of computation that must be performed to

generate cells at the source LPs. Because simulator events contain relatively little computation, the overhead for scheduling and processing events significantly degrades the performance of the fine-grained simulator, which must schedule a new event for each source each clock tick.

A significant performance degradation was observed in the coarse-grained simulator as the number of processors is increased. This is because use of large LPs makes the simulator more sensitive to rollbacks, a phenomenon observed in the experiments using synthetic workloads. Figure 5 shows the fraction of processed events that are rolled back. As noted earlier, rolling back a large LP introduces unnecessary rollbacks in the computation, which in turn can result in unnecessary anti-messages and additional rollbacks in other LPs. We observed that both the small- and large-grained LPs yielded more rollbacks as the number of processors increased, as would be expected, however, the increased number of rollbacks was far more pronounced in the coarse-grained simulation. This ultimately led to the fine-grained simulation outperforming the coarse grained simulation, although the fine grained simulator using up to four processors never outperformed the coarse grained simulator using only two processors. It is possible that the fine grained simulation could yield the best performance overall if more processors were available for these experiments.

A second factor affecting performance with four processors is the presence of operating system daemons running on one processor of the four. The processor mappings assume identical processors, and thus allocate processing equally. We observed more severe performance degradations for the small-LP simulation than the large-LP simulations when all four processors were used (The experiments with fewer processors avoid the one executing the daemons). While in principal one could isolate the machine to avoid such interference, such a stand-alone configuration is not what one would typically encounter in practice. A more pragmatic approach is to adjust the processor mapping to compensate for this effect. Further experiments are are required to study this question, and to optimize performance of both simulators for increased numbers of processors.

Clustering mapping yields the best performance because it reduces the amount of inter-processor communications. This partitioning algorithm also yielded fewer rollbacks. The balanced random mapping yielded better performance than the simple random mapping. Balancing the workload reduces the number of rollbacks, as one would expect. Overall, simple mapping strategies such as random distribution of LPs to processors yielded poor performance for both the large and small grained simulators, especially as the number of processors was increased.

6.2 Experiments using State Updates

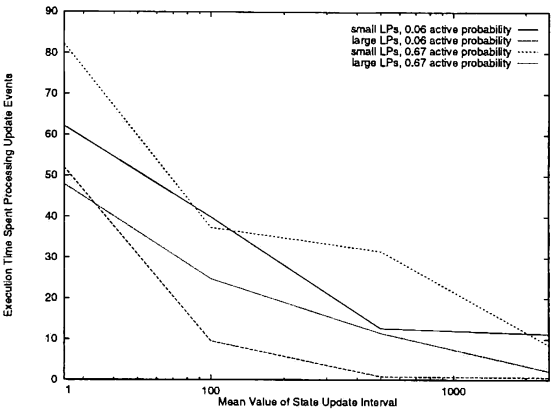

Figure 6: Time spent processing update events

A second set of experiments were performed to evaluate the effect of state updates on the relative performance of the two simulators. The same traffic generation model was used as in the experiments without state updates. As mentioned earlier, state updates were generated within each switch with an exponentially distributed time between updates. Experiments used 25, 100, 500, and 2500 as the mean values of the state update interval, where five units corresponds to the amount of time required to transmit one cell over a link. Each set of experiments were executed on one to four processors using the clustering approach to network partitioning.

The time spent performing state updates of the two simulator on four processors is shown in Figure 6; similar data was observed for different numbers of processors. The execution time of the small-grained simulation without state updates was 92 and 171 seconds for the lightly and heavily loaded networks, respectively. Execution times for the large-grained simulator were 59 and 116 seconds, again for the lightly and heavily loaded networks, respectively. As expected, as the state of each switch is updated more frequently (moving to the left in Figure 6), the fine-grained simulator incurs more state update overhead than the coarse grained-simulator. This overhead represents a significant portion of the simulation computation at high update frequencies. It is perhaps noteworthy that the performance of the fine-grained simulator would be degraded even further if LPs modeling a single switch were distributed across multiple processors because interprocessor communication would

then be required with each state update.

7 CONCLUSIONS AND FUTURE WORK

This study addressed the question of partitioning simulations into logical processes, and whether small-grained or large-grained LPs yielded better performance. For the test cases that were examined, the central conclusions from this study are: (1) The most dominant effect favoring large-grained logical processes in these experiments was their ability to combine events containing the same timestamp into a single event. (2) Large grained logical processes suffered from being relatively sensitive to rollback as the number of processors increased, especially when the workload is low. Unnecessary rollbacks due to aggregating too many components into a single LP can have significant, detrimental effects in Time Warp simulations. (3) Partitioning the simulation to avoid state variables that are shared between LPs results in somewhat higher overheads for state updates, particularly if the state variables are updated very frequently.

Work is continuing to collect additional data across a variety of other test cases, e.g., other network topologies, source traffic models, and traffic loads. with the goal of developing an efficient, flexible parallel simulation tool for modeling ATM networks. Simulations using small sized LPs are able, in principal, to achieve better load balance because portions of a switch may be distributed over more than one processor, an aspect not explored in the results described here.

ACKNOWLEDGEMENTS

This work was supported by NSF Grant Number CDA-9501637 and NCR-9527163.

REFERENCES

Bae, J., and T. Suda. 1991. Survey of traffic control schemes and protocols in ATM networks. *Proceedings of the IEEE*, 79(2).

Das, S., R. Fujimoto, K. Panesar, D. Allison, and M. Hybinette. 1994. GTW: A Time Warp system for shared memory multiprocessors. In *1994 Winter Simulation Conference Proceedings*, pages 1332–1339.

Davoren, M., 1989. A structural mapping for parallel digital logic simulation. In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 21, pages 179–182. SCS Simulation Series.

Earnshaw, R. W., and A. Hind., 1992. A parallel simulator for performance modelling of broadband telecommunication networks. In *1992 Winter Simulation Conference Proceedings*, pages 1365–1373.

Fujimoto, R. 1990a. Performance of Time Warp under synthethic workloads. In *Distributed Simulation*, volume 22, pages 23–28. SCS Simulation Series.

Fujimoto, R. M. 1990b. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53.

Gaujal, B., A. G. Greenberg, and D. M. Nicol. 1993. A sweep algorithm for massively parallel simulation of circuit-switched networks. *Journal of Parallel and Distributed Computing*, 18(4):484–500.

Jefferson, D. R. 1985. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425.

Nandy, B., and W. Loucks. 1992. An algorithm for partitioning and mapping conservative parallel simulation onto multicomputers. In 6^{th} *Workshop on Parallel and Distributed Simulation*, volume 24, pages 139–146. SCS Simulation Series.

Nicol, D. M., and R. M. Fujimoto. 1994. Parallel simulation today. *Annals of Operations Research*, 53:249–286.

Phillips, C. I., and L. G. Cuthbert. 1991. Concurrent discrete-event simulation tools. *IEEE Journal on Selected Areas in Communications*, 9(3):477–485.

Tallieu, F., and F. Verboven. 1991. Using Time Warp for computer network simulations on transputers. In *Proceedings of the 24th Annual Simulation Symposium*, volume 21, pages 112–117. IEEE Computer Society Press.

Turner, S., and M. Xu. 1992. Performance evaluation of the bounded Time Warp algorithm. In 6^{th} *Workshop on Parallel and Distributed Simulation*, volume 24, pages 117–128. SCS Simulation Series.

Unger, B., and Z. Xiao. 1994. The fast parallel simulation of telecommunication networks. In *Proceedings of New Directions in Simulation for Manufacturing and Communications*. The Operations Research Society of Japan.

Wieland, F., and D. R. Jefferson. 1989. Case studies in serial and parallel simulation. In *Proceedings of the 1989 International Conference on Parallel Processing*, volume 3, pages 255–258.

AUTHOR BIOGRAPHIES

FANG HAO is a Ph.D. student in the College of Computing at Georgia Tech.

KAREN WILSON received the M.S. degree in Computer Science at Georgia Tech in 1995. She is currently employed by Digital Equipment Corporation.

RICHARD FUJIMOTO is a professor in the College of Computing at Georgia Tech.

ELLEN ZEGURA is an assistant professor in the College of Computing at Georgia Tech.