Proceedings of the 1995 Winter Simulation Conference
ed. C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman

A COMPARISON OF TWO METHODS FOR ADVANCING TIME IN PARALLEL DISCRETE EVENT SIMULATION

Anthony P. Galluscio

Software Technology Inc.
5904 Richmond Highway
Suite 610
Alexandria, Virginia 22303, U.S.A.

John T. Douglass
Brian A. Malloy
A. Joe Turner

Department of Computer Science
Clemson University
Clemson, South Carolina 29634, U.S.A.

ABSTRACT

We compare the design and implementation of a parallel simulation of a traffic flow network using two different approaches: event-driven and time-driven. Our experiments with the sequential implementation of the two approaches correlates with previous research (Nance 1971). We design a conservative parallel implementation of the traffic flow problem where we obtain a maximum speedup of 9.27 using 16 Sun workstations running under *parallel virtual machine* or PVM (Geist et al. 1993). We use wall-clock time as a measure of execution speed. We show that appreciable speedup can be achieved in parallelizing either the event-driven or time-driven approach. We also show that speedup is a misleading metric when used to compare the parallelizability of the two approaches. Parallel performance, as measured by speedup, may be better when the sequential performance is poor. For example, the time-driven implementation achieved better speedup than the event-driven implementation for few cars in the system; however the sequential time-driven implementation required longer to execute than the event-driven implementation for few cars in the system. Similarly, for many cars in the system, the event-driven implementation achieved better speedup than the time-driven implementation.

1 INTRODUCTION

In this paper, we present the design and implementation of a traffic flow network using two different approaches: event-driven and time-driven (Nance 1981). We begin by designing an efficient event-driven approach to model the traffic flow network. Our design of the event-driven traffic model matches the design of a time-driven traffic model (Douglass and Malloy 1994). A parameter to the models is *traffic flow*, a measure of the average number of cars entering the system for each clock tick. In the sequential execution of the two models for sparse traffic flow, the event-driven implementation ran faster than the time-driven model. For dense traffic flow patterns, the sequential execution of the time-driven model ran faster than the event driven model. This result correlates with previous research (Nance 1971).

Using previously developed techniques to exploit the look-ahead in the traffic model (Douglass and Malloy 1994), we parallelize our event-driven implementation of the traffic flow model. We use the conservative protocol (Fujimoto 1990) for asynchronous execution using distributed local clocks. For the parallel executions, we obtain a maximum speedup of 9.27 using 16 Sun workstations. This speedup is appreciable since our parallel architecture is *parallel virtual machine* or PVM (Geist et al. 1993), not known for fast communication, and our processes ran in the background using the Unix facility "nice". Also, we use wall-clock time as a measure of execution speed. We then compare the parallel implementation of the event-driven approach to the parallel implementation of the time-driven approach. We draw some interesting conclusions about the parallelizability of the two models.

In the next section we discuss background for our work including a discussion of PVM and the conservative approach to parallelizing simulation. In section 3, we overview the event-driven and time-driven approach, followed by the results of our experiments in 4. In section 5, we draw conclusions.

2 BACKGROUND

The protocols used to design and implement parallel simulation programs fall broadly into two categories: conservative and optimistic. We begin this section with a brief overview of these two protocols. We then present the features of PVM, the software package that we use to construct a parallel machine using a

network of Sun workstations.

2.1 Protocols for Parallel Simulation

The protocols currently used to parallelize a simulation program fall into two general categories: conservative and optimistic. Excellent surveys of these approaches can be found in Fujimoto (1990) and Righter and Walrand (1989). In conservative simulations, a processor does not execute an event e at simulation time t_e until all messages with time stamp less than t_e have been processed. This sequencing of events is known as the *local causality constraint* and strict adherence to this constraint guarantees that the execution of event e is correct. There are numerous conservative algorithms that differ primarily in the manner in which they adhere to this constraint.

In optimistic simulations, a processor may execute events in any order and violations of the local causality constraint are corrected by *rolling back* the processor to a state where the constraint holds. The optimistic approach, such as Time Warp (Jefferson 1985), can produce substantial speed up due to parallelism (Madisetti and Hardaker 1992, Fujimoto 1990, Baezner, Rohs, and Jones 1992, Unger et al. 1990). An excellent variation of the Time Warp approach can be found in Madisetti, Walrand and Messerschmitt (1988).

2.2 Parallel Virtual Machine

Parallel Virtual Machine (PVM) (Geist et al. 1993) is a software package that provides support for the construction of a parallel computer using a network of workstations. PVM supports a message passing communication paradigm that can accommodate more than 25 platforms, ranging from a Cray/YMP to an 80386 personal computer running the Unix operating system. Messages may be passed between any of the machines supported; data conversions, for platforms that use different data representations, are transparent to the user.

3 OVERVIEW OF THE MODELS

In any simulation, it is essential that the important details of the system under study are captured by the model being designed. To make a fair comparison of the event-driven and time-driven approach, both models are designed to capture the important details of the traffic flow problem that we study. In this section, we show how the two models represent traffic flow in a very different fashion yet successfully capture the important details of the system with respect to the movement of cars through the network. For details of our technique to model contention in the traffic network, see Galluscio and Malloy (1995).

3.1 Representation of the Traffic Network in the Two Models

For both models, we represent the traffic network as a square mesh composed of streets running in the horizontal and vertical directions with traffic lights at the intersections of the streets. The square mesh is a flexible representation that is easily modified to produce different size workloads; this flexibility enables us to investigate the effects of increasing the workload on the parallelized implementations of the two models. In the parallelized version of the network, the square mesh of streets and lights is partitioned into subsystems called *grids* where each grid is assigned to a processor. Figure 1 illustrates a traffic network composed of two grids where each grid contains four lights. The figure illustrates the logical view of the network which is the same for both models; however, the actual implementation of the network is different in each approach.

In the event-driven approach, the simulation model can be viewed as a model of the interaction of discrete events occurring in the system. For example, arriving at an intersection and departing an intersection become events in the model, where each pending event is in an event queue and system time is the time-stamp of the currently executing event. This representation is illustrated at the top of Figure 3.

In the time-driven approach, the activities in the model are scanned on each clock cycle to determine if a state change can occur in the model. Intersections are logical structures in the model, in the implementation each intersection is represented by its corresponding queue data structure as illustrated at the bottom of Figure 3. As the simulation progresses, the *list of activities* is scanned so that a car enters a street by being inserted into the corresponding queue and the car enters an intersection by being inserted into the service queue for that intersection. There is no event queue in our time-driven model. Thus, entering or departing a street is an $O(1)$ queue operation in the time-driven approach; in the event-driven approach, entering or departing a street is an $O(\log\ n)$ operation on a priority queue where n is the number of events in the priority queue.

From a logical perspective the models are identical in the way cars enter the simulation and travel within and between grids. A car enters the simulation at a construction site labeled SOURCE_SINK. All of the boundary streets illustrated in Figure 1 are sources and sinks, where cars are generated according to a

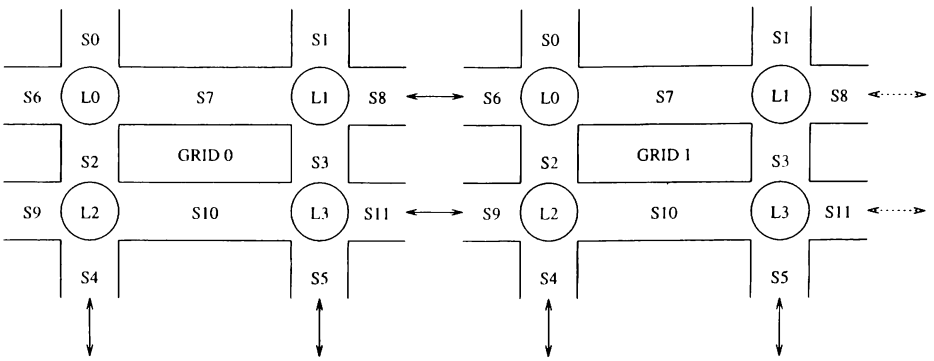

Figure 1: A traffic network composed of two grids with each grid containing four lights and twelve street segments. In the parallel simulation, each of the grids is assigned to a processor.

fixed probability requirement. Cars may travel either north, south, east, or west with a fixed probability of changing direction at any given intersection. Contention at the intersections is taken into account. For example, if a car is turning left into the path of a car going straight, then a contention mechanism inhibits one of the cars until the other car clears the intersection (see Galluscio and Malloy 1995, for a detailed description of the contention mechanism). Both models allow cars to travel between grids using message passing in the distributed PVM environment. Cars traveling the same direction on a grid are collected and sent to the destination grid upon expiration of the timing mechanism that is used to exploit lookahead. The next section reveals any inherent differences in the approaches.

3.2 Simulating Traffic Flow in the Two Models

Valid conclusions about the relative parallelizability of the simulation approaches require a fair comparison of the two models, which in turn relies on a similar representation of traffic. Although the models represent traffic similarly, the simulation approaches and the underlying algorithms differ dramatically. The rest of this section describes the time-driven algorithm, describes the event driven algorithm, and draws distinctions between the two approaches.

Figure 2 illustrates the general algorithm used to simulate the traffic network for the time-driven model. The algorithm starts with a local time of zero and iterates until the local time is equal to the input maximum time. The local time is incremented by one tick each time through the main loop. In the main loop the algorithm first processes null messages and car messages from other processors. The

```
algorithm   SimulateTraffic
input       MaxSimTime
output      Simulation of this grid

begin SimulateTraffic
   LocalTime = 0
   while LocalTime <= MaxSimTime loop
        while more null messages in receive buffer loop
             process the null message
        end while
        while more car messages in receive buffer loop
             process the car message
        end while
        if not violating local causality constraint then
             update lights
             process segments by
                  1. gen cars for source
                  2. consume cars for sink
                  3. pass cars to neighboring processors
                  4. move cars to adjacent segments
             increment local time
        end if
   end while
end
```

Figure 2: General algorithm for parallel simulation of a traffic flow network using the time-driven approach.

null messages must be processed early in the loop to preserve the local causality constraint. The local causality constraint dictates the next action. If it is possible to proceed without violating the local causality constraint, then the algorithm updates lights and processes street segments. Processing the street segments involves generating new cars at the sources, consuming old cars at the sinks, passing cars to neighboring processors, and moving cars to adjacent local segments. Two points of interest are that the clock always increments by a single tick, and that cars move

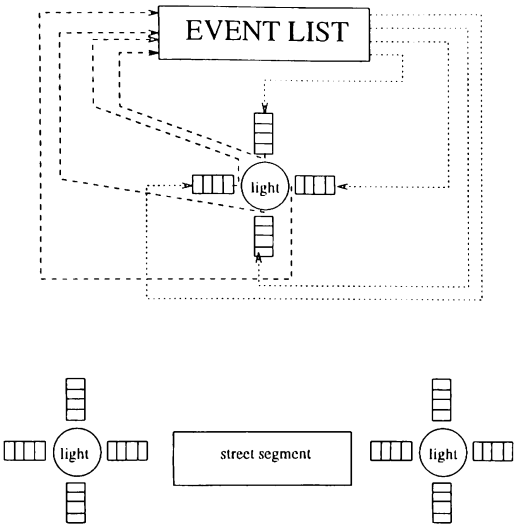

Figure 3: The representation of street segments for the two traffic flow models. The model at the top of the figure is the event-driven approach where cars traverse the traffic network by being processed as events. The model at the bottom is the time-driven approach where cars traverse the traffic network by being inserted or removed from a queue.

directly from street segment to street segment.

The *event* is the touchstone characteristic of the event-driven approach. The allure of the event driven model is its extensibility; extensions to an event-driven simulation are handled by adding events. The algorithm for simulating the parallel event-driven model is provided in Figure 4. The algorithm uses an event_list, a main loop, and five distinct event-types. The main loop iterates until the local time is greater than or equal to the maximum simulation time. The first action of the main loop is to remove the event with the minimum time stamp from the event_list, and update the local clock to the time stamp of that minimum event. The minimum event is then processed according to its event type. Processing the minimum event may cause other events to be inserted into the event_list. Note that the clock may increment by an arbitrary amount corresponding to the time stamp of the minimum event.

Differences between the event-driven and time-driven approaches arise in the way time is handled and the way cars move through the simulation. Those differences impact the speed and flexibility of the simulation techniques. The event-driven approach increments the clock by an arbitrary amount that corresponds to the minimum time-stamped event on the current event list. Therefore, large *jumps* in time are expected for sparse traffic networks. On the other hand, the time-driven approach checks for new tasks to perform after every tick. Unproductive checking would be expected for sparse traffic flow networks. Another difference between the approaches is the way simulated cars proceed through the simulation. The time-driven approach moves simulated cars from street to street using $O(1)$ queue operations; however, the event-driven approach uses an event list, implemented as a priority queue, and moves simulated car events in and out of the event list using $O(\log\ n)$ heap operations. As a result, the sequential time-driven approach may be much faster for a packed traffic network, and only moderately faster for a sparse traffic network. From our experience the event-driven model is more easily extended and enhanced.

Table 1: Summary of the statistics computed by the event-driven and time-driven models when executed on 16 processors. These statistics were gathered for the traffic model with a workload of 576 lights and a traffic flow of 3.5 million cars in the system.

Statistic	*cars generated*	*cars exiting*	*total messages*
Event	3,451,303	3,438,014	479,952
Time	3,359,579	3,345,447	479,921

```
algorithm   SimulateTraffic
input       Futureeventslist, travel_time (10),
            crossing_time (2), and next_light (5)
output      simulation time
events      ARRIVE, LEAVE, LIGHT, RECEIVE, SEND

begin SimulateTraffic
   while TIME <= max_time loop
         event = get_event(event_list)
         TIME = event.time
         case event.type
               ARRIVE: process_arrival
               LEAVE: process_leave
               LIGHT: process_light
               RECEIVE: process_receive
               SEND: process_send
         end case
   end while
end SimulateTraffic
```

Figure 4: General algorithm for parallel simulation of a traffic flow network using the event-driven approach.

4 EXPERIMENTAL RESULTS

Both the event-driven and the time-driven models are parameterized, where the parameters describe the probability that a car will turn, the length of the simulation, the numbers of processors, the *light interval*, the probability that a car is generated at a source, and the number of lights (and streets) in the model. We define *light interval* as a triple (r, g, y), where r is the number of clock ticks that the light is red, g is the number of clock ticks that the light is green, and y is the number of clock ticks that the light is yellow. To facilitate our comparison of the approaches, we keep the first four parameters fixed and vary the final two parameters.

We use the fifth parameter, the probability that a car is generated at a source, to control traffic flow: the *denseness* or *sparseness* of the traffic as it flows in the network. We vary this parameter from 0.00001 to .35 where the first value results in an average of 100 cars being generated during the simulation and the second value results in an average of 3.5 million cars being generated during the simulation.

We use the final parameter, the number of lights (and streets) in the model, to control workload. We vary this parameter in our experiments so that the number of lights is a perfect square ranging from 16 to 576 lights in the traffic network. Increasing the number of lights induces a corresponding increase in the number of streets, which results in a greater number of cars in the system.

This section begins by presenting summary statistics to show that the implementations of the two models are similar in their representations of the simulated traffic network. We then present the results obtained by varying the parameters that describe traffic flow and workload.

4.1 Summary Statistics

The summary statistics in Table 1 illustrate that the implementations of the event-driven and time-driven models represent the same traffic network. The summary was garnered by executing each of the parallel implementations on a network of 16 Sun SLC workstations. The data in Table 1 represents averages over 30 executions of the simulation with a *traffic flow* of 3.5 million cars, or an average of 3.5 cars per street on a simulation grid, and a workload of 576 lights or 36 lights in the system.

The first column in Table 1, *cars generated*, illustrates that both models generated similar numbers of cars along the border of the network with 3,451,303 cars generated for the event-driven approach and 3,359,579 cars generated for the time-driven approach. The two numbers in this first column differ by one percent.

The second column in Table 1, *cars exiting*, indicates that similar numbers of cars exited both systems with 3,438,014 cars exiting for the event-driven approach and 3,345,447 cars exiting for the time-driven approach. The two numbers in this second column differ by one percent.

The third column in Table 1, *total messages*, indicates that both models generated similar numbers of messages during the parallel simulation with 479,952 messages generated in the event-driven approach and 479,921 messages generated in the time-driven approach. The two numbers in this third column differ by one percent. There are two types of messages generated during the executions: *null messages* and *car messages*. Null messages are required in the conservative approach to avoid deadlock.

4.2 The Effect of Increased Traffic Flow on the Two Models

Figure 5 illustrates the effect of increased traffic flow on the execution time of the two models. For the graph, the vertical axis indicates execution time in minutes and the horizontal axis indicates traffic flow as it increases from an average of 100 cars in the simulation to 3.5 million cars in the simulation, where each simulation runs for 100,000 clock ticks.

For the graph of Figure 5, the solid line (diamond) and the dashed line (plus sign) compare the sequen-

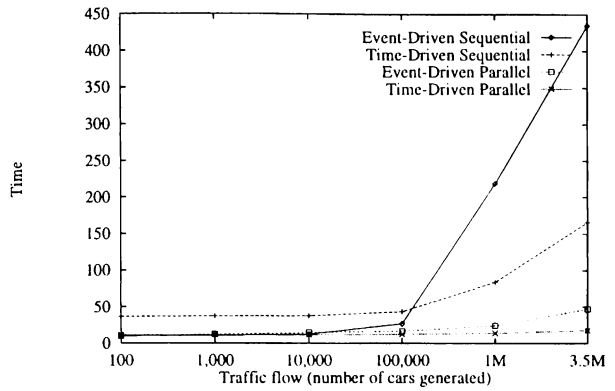

Figure 5: This figure illustrates the effects of increasing traffic flow density on the average execution time of the event-driven implementation and the time-driven implementation. For the experiments, traffic flow is increased from an average of 100 cars to 3.5 million cars in the system. The number of lights in the traffic network is held constant at 576 lights for all executions. Time is given in minutes and the parallel executions were run on a network of 16 SLC Sun workstations.

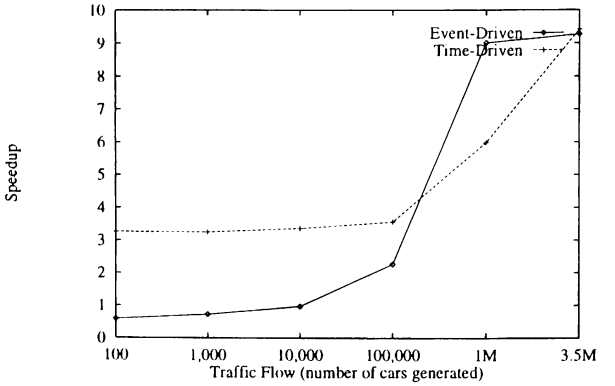

Figure 6: Comparison of speedup for varying densities of traffic flow in the network.

tial execution times of the models; these lines illustrate the behavior of the two models for sparse and dense traffic flow and they correlate with previous results (Nance 1971). For sparse traffic flow, where the numbers of cars generated varied from 100 to 100,000 cars, the event-driven implementation executed faster than the time-driven implementation. For dense *traffic flow* where the numbers of cars generated varied from 100,000 to 3.5 million, the time-driven implementation executed faster than the event-driven implementation.

In addition to the relative speed of the sequential implementations of the time-driven and event-driven approaches, Figure 5 also indicates the number of cars in the simulation at the break-even point. The *break-even* point in the comparison of the sequential executions represents the number of cars required so that the running time of the two implementations is the same. Consider that, in the time-driven implementation, a list of 2304 events (576 lights x 4 service queues at each light) must be scanned on every clock tick, regardless of the time of the next event. Thus, on each clock tick, the running time of the time-driven approach is 2304 $O(1)$ queue operations plus the time to process the events in the queues that are ready to be processed.

The event-driven implementation is not required to process events at every clock tick but can advance time to that of the next scheduled event. As Figure 5 indicates, the *break-even* point occurs in the system that contains 100,000 cars over the execution time. Since the two implementations ran for 100,000 clock ticks and 100,000 cars are generated in each simulation, an average of one car is generated at each tick. For traffic flow densities less than 100,000 cars, the event-driven implementation ran faster than the time-driven implementation because it can increment simulated time to match the time of the next scheduled event.

Figure 5 further indicates that for the sparsest traffic flow of 100 cars, the event-driven implementation ran 3.7 times faster than the time-driven implementation and for the densest traffic flow of 3.5 million cars, the time-driven implementation ran 2.6 times faster than the event-driven implementation. A similar pattern of execution times is illustrated for the parallel execution of the two models. In the parallel simulations for sparse traffic flow, the event-driven implementation ran faster (9.9 minutes) than the time-driven implementation (11.3 minutes). For dense *traffic flow*, the time-driven implementation ran faster (17.6 minutes) than the event-driven implementation (46.9 minutes). Thus, if execution speed is the prime concern, the event-driven implementation performs better for sparse traffic flow and the time-driven implementation performs better for dense traffic flow.

However, the pattern illustrated above for **execu-**

tion time is reversed when considering the **speedup** achieved in the parallel executions of the two models. Figure 6 compares speedup for varying densities of traffic flow ranging from 100 to 3.5 million cars generated during the simulation. The implementations of the time-driven approach run slower than the implementations of the event-driven approach for sparse *traffic flow*. However the time-driven implementation achieves better speedup than the event-driven implementation for sparse traffic flow; this can be seen in Figure 6 where the time-driven implementation (solid line in the graph) shows larger speedup than the event-driven implementation (dashed line in the graph). This higher speedup for the time-driven implementation for sparse *traffic flow* results from the slower sequential execution speed, since this results in a higher ratio when computing speedup.

4.3 The Effect of Increased Workload on the Two Models

In this section we present the results of our comparison of the two models with different levels of workload. Figure 7 illustrates the effect of increased workload on the execution time of the two models. For the graph, the vertical axis indicates execution time in minutes and the horizontal axis indicates the number of lights in the system as they increase from 16 to 576 lights. All data was collected with the traffic flow density held constant at one million cars in the system.

A comparison of Figure 7 and Figure 5 illustrates that varying traffic flow and workload has similar effects on execution time. When the traffic network contains few lights, the event-driven implementation executes faster than the time-driven implementation; when the traffic network contains many lights, the time-driven implementation executes faster than the event-driven implementation.

Figure 8, illustrates that increasing the workload, improves speedup. Previous research has shown that when communication is expensive, increasing the amount of work performed by each processor can offset the communication cost and result in improved speedup (Carothers et al. 1994, Douglass and Malloy 1994) The graph in Figure 8 illustrates that the increase in speedup was similar in both models.

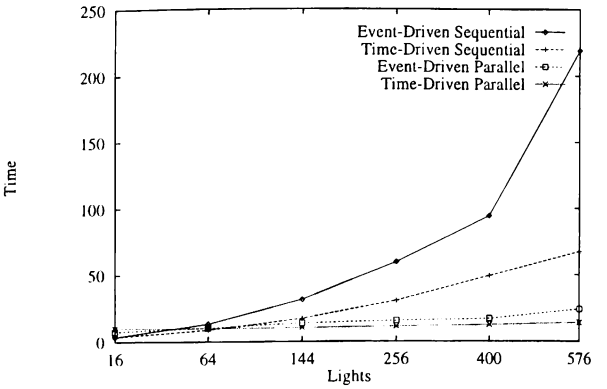

Figure 7: The graph in this figure illustrates the impact of workload on the average running time of the simulation.

5 CONCLUSIONS

In this paper, we have presented the design and implementation of a parallel simulation of a traffic flow network using two different approaches: event-driven and time-driven. Our experiments with the sequential implementation of the two approaches correlates with previous research (Nance 1971).

We have shown that, for the traffic network simulation, implementations of both the event-driven approach and the time-driven approach can achieve appreciable speedup. This speedup can be achieved in a distributed parallel environment using a parallel architecture such as PVM, which extracts high cost for communication.

We have also shown that speedup is a misleading metric when used to compare two models. For example, for dense traffic flow in the network, the implementation of the time-driven approach achieved less speedup than the implementation of the event-driven approach, yet the parallel time-driven implementation executed faster (17.6 minutes) than the parallel event-driven implementation (46.9 minutes).

ACKNOWLEDGEMENTS

The idea of comparing the two approaches to simulation was suggested by Richard E. Nance. Furthermore, many of the ideas that permeate this paper are derived from studying his work.

REFERENCES

Baezner, D., C. Rohs, and H. Jones. 1992. U. S. Army MODSIM on Jade's Time Warp. *Proceedings of the 1992 Winter Simulation Conference* 665–671.

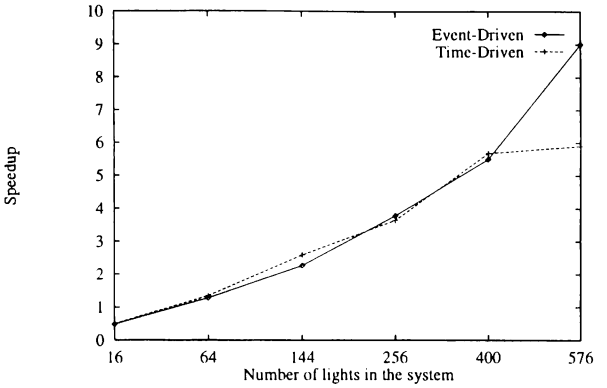

Figure 8: Graphical comparison of speedup for varying numbers of lights in the system. All executions used a network of 16 workstations so that for 16 lights in the system, one light is assigned to each processor. For 576 lights in the system, 24 lights are assigned to each processor.

Carothers, C. D., R. M. Fujimoto, Y. B. Lin, et al. 1994. Distributed Simlation of Large Scale PCS Networks. *Proceedings of the Second International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* 2–6.

Douglass, J. and B. Malloy. 1994. Using a Shot Clock to Design an Efficient Parallel Simulation. *Proceedings of the 1994 Winter Simulation Conference* 1362–1369.

Fujimoto, R. M. 1990. Parallel Discrete Event Simulation. *Communications of the ACM* 33(10):31–53.

Galluscio, A. and B. Malloy. 1995. The Event-Driven and Time-Driven Approach to Parallel Simulation: A Comparison. Technical Report #95-105, Clemson University.

Geist, A., A. Beguilin, J. Dongarra, et. al. 1993. PVM 3 User's Guide and Reference Manual. *Oak Ridge National Laboratory* ORNL/TM-12-87.

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

Lubachevsky, B. 1988. Bounded Lag Distributed Discrete Event Simulation. *Proceedings of the 1988 SCS Multiconference on Distributed Simulation* 183–191.

Madisetti, V., and D. Hardaker. 1992. Synchronization Mechanisms for Distributed Event-Driven Computation. *ACM Transactions on Modeling and Computer Simulation* 2(1):12–51.

Madisetti, V., J. Walrand, and D. Messerschmitt. 1988. WOLF: A Rollback Algorithm for Optimistic Distributed Simulation Systems. *Proceedings of the 1988 Winter Simulation Conference* 296–305.

Nance, R. E., 1981. The Time and State Relationships in Simulation Modeling. *Communications of the ACM* 24(4):173–179.

Nance, R. E., 1971. On Time Flow Mechanisms for Discrete System Simulation. *Management Science* 18(1):59–73.

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

Unger B. W., J. Cleary, A. Dewar, and Z. Xiao. 1990. A Multi-Lingual Optimistic Distributed Simulator. *Transactions of the Society for Computer Simulation* 7(2):121–152.

AUTHOR BIOGRAPHIES

ANTHONY P. GALLUSCIO is a computer scientist at Software Technology Inc., where he designs real time software systems. He received an M.S. from Clemson University in 1995.

JOHN T. DOUGLASS is a graduate student in the department of Computer Science at Clemson University. He received an M.S. from Clemson University in 1994. His research interests include program analysis, and language design techniques for parallelism.

BRIAN A. MALLOY is an Assistant Professor in the department of Computer Science at Clemson University. He received an M.S. and Ph.D. from the University of Pittsburgh in 1984 and 1991 respectively. His research interests include compilation techniques for parallelism, parallel discrete event simulation and language design techniques for simulation.

A. JOE TURNER Joe Turner is a Professor of Computer Science at Clemson University. His research interests include software engineering and computer science education.