A Comparison of Two Methods for Advancing Time In Parallel Discrete Event Simulation

Anthony P. Galluscio, John T. Douglass
Brain A. Malloy and A. Joe Turner
malloy@cs.clemson.edu
Department of Computer Science
Clemson University
Clemson, SC 29634

Abstract

In this paper, we compare the design and implementation of a parallel simulation of a traffic flow network using two different approaches: event-driven and time-driven. We begin by designing an efficient event-driven approach to model the traffic network; our design matches a time-driven model of the same traffic network[5]. Our experiments with the sequential implementation of the two approaches correlates with previous research [12]. Exploiting the look-ahead in the event-driven model, we design a conservative parallel implementation of the traffic flow problem where we obtain a maximum speedup of 9.27 using 16 Sun workstations. This speedup is appreciable since our parallel architecture is parallel virtual machine (PVM)[8], not known for fast communication, and 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 (3.21 to 3.56) than the event-driven implementation (0.59 to 0.97) 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 (9.01 to 9.27) than the time-driven implementation (5.99 to 9.12).

keywords: modeling, event-driven, time-driven, methodology, conservative parallel distributed simulation.

1 Introduction

There are three accepted approaches to the design of a simulation model: the event-driven approach, the time-driven approach and the process-driven approach. Each of these approaches can be distinguished by the method used to control the passage of time. In the event-driven approach, time is dynamically updated to reflect the time of the next scheduled event in the model. In the time-driven approach, a list of activities is examined on each timing cycle to determine if a state change can occur; time is updated at the end of each cycle. In the process-driven approach, the actions of each object in the model are incorporated into a corresponding process that is then scheduled for execution; time in the process-driven approach is updated to reflect the time of the currently active process in the model. The approach chosen, when designing a model, should be based on the characteristics of the system under study. For example, having time progress in ticks may provide more control for such applications as on-line graphics or other applications where events happen regularly and in fixed increments[13]. However, it may be difficult to determine, in advance, the frequency of event occurrence in the simulated system.

A further problem is that the sequential execution of the simulation may be so computationally intensive that the result is not computed in a timely manner. Thus, one solution is to parallelize the simulation to speed up the execution. Among the approaches, one may produce better speedup than another when parallelized. However, no research has produced evidence that any of the three approaches result in better speedup than another when parallelized.

In this paper, we present the design and implementation of a traffic flow network using two different approaches: event-driven and time-driven. 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 [5]. 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 [12].

Using previously developed techniques to exploit the look-ahead in the traffic model[5], we parallelize our

event-driven implementation of the traffic flow model. We use the conservative protocol 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 (PVM)[8], 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.

We first conclude that appreciable speedup can be achieved in parallelizing either the event-driven or the time-driven approach. We varied the *traffic flow* in each model so that the number of cars generated in the system varied from 100 cars to 3.5 million cars; the simulation ran for 100,000 clock ticks on 16 workstations. Using various levels of *traffic flow*, speedup for the event-driven approach ranged from 0.59[1] to 9.27, and speedup for the time-driven approach ranged from 3.21 to 9.42. Thus, for sparse levels of traffic, the time-driven implementation provides better speedup than the event-driven implementation

Our second conclusion is that speedup can be a misleading metric when used to compare two different 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 (3.21 to 3.56) than the event-driven implementation (0.59 to 0.97) for sparse *traffic flow*; however, as stated above, the sequential time-driven implementation required longer to execute than the event-driven implementation for sparse *traffic flow*. Similarly, for dense levels of *traffic flow*, the event-driven implementation achieved better speedup (9.01 to 9.27) than the time-driven implementation (5.99 to 9.12).

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 section 5, we draw conclusions.

[1] Speedup value less than one indicates that the parallel execution required longer to complete than the sequential execution.

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 [6] and [13]. 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[9], has shown to produce substantial speed up due to parallelism [10, 6, 1, 14]. An excellent variation of the Time Warp approach can be found in [11].

2.2 Parallel Virtual Machine

Parallel Virtual Machine (PVM) [8] is a software package[2] 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 which use different data representations, are transparent to the user.

There are two communication protocols supported by PVM, allowing the programmer to choose between *dynamic TCP sockets* or *UDP communication through the PVM daemon.* The default communication protocol is UDP communication. In UDP communication, user processes make PVM library calls. The PVM

[2]available through anonymous ftp from netlib2.ornl.gov in the pvm3 directory

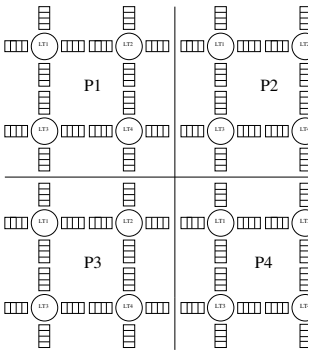


Figure 1: A traffic network composed of four grids with each grid containing four lights and twelve streets. In the parallel simulation, each of the grids is assigned to a processor.

daemon receives this information and mediates the communication. In the dynamic TCP socket communication, the user makes the same library calls as in the UDP approach, however on the first communication between two processes, a TCP socket is established by daemons running on each of the machines. Once established, this socket is used for all subsequent communication between the two processes so that the daemons are not involved in the mediation. The initial communication using the dynamic TCP sockets is significantly more expensive than subsequent TCP communication or any UDP communication because of the overhead incurred to establish the sockets. However, subsequent TCP communication is far less expensive than UDP communication; thus, if communication occurs many times over the course of a program than TCP socket communication is significantly more efficient than UDP communication. The results of previous experimentation demonstrate the savings of repeated TCP communication [5].

The cost of the communication in PVM, regardless of the protocol used, is high. Since PVM is running on a network of machines the time needed to pass a message is many times that of messages being passed in a dedicated multiprocessor. Reducing communication in programs running in PVM is therefore a crucial consideration.

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

```
algorithm    ConstructLocalGrid
input        Streets, an array of street records
output       Initialized local grid

begin ConstructLocalGrid
    Determine L, the number of lights
    for i = 1 to L loop
        for each direction N, S, E, W loop
            St_i = get_street_num(direction, i)
            if street[St_i] is not marked
                street[St_i].inbound=create_Q()
                street[St_i].outbound=create_Q()
                if street[St_i] connects to another processor
                    street[St_i].construction = COM_LINK
                else
                    street[St_i].construction = SOURCE_SINK
                end if
            else /* This street has been visited before */
                street[St_i].construction = PASS_WITHIN
            end if
            Mark street[St_i] as visited
        end for
    end for
end ConstructLocalGrid
```

Figure 2: Algorithm to construct a grid for the parallelized model. In the parallel simulation, each grid will be local to a processor.

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[7].

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 direction with traffic lights at the intersection 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 four 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. Thus, traveling along a street, 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

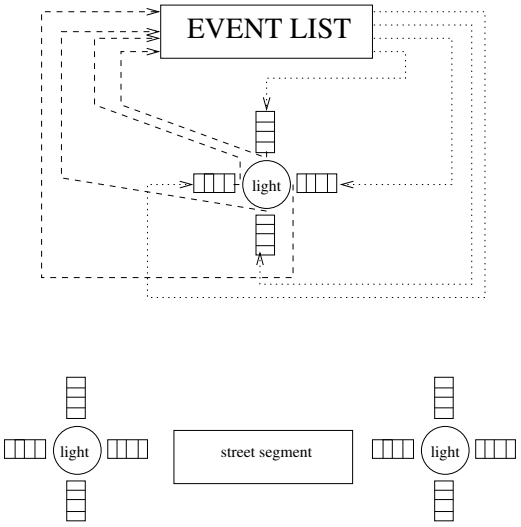


Figure 3: The representation of streets for the two traffic flow models. The model at the top of the figure is the event-driven approach where cars traverse the traffic network by being processed as events. The model at the bottom is the time-driven approach where cars traverse the traffic network by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

---

is the time-stamp of the currently executing event; this representation is illustrated at the top of Figure 3. Travel along a street, a logical structure, is an event.

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. Streets are logical structures in the model; however, in the implementation, each street is represented by a corresponding queue data structure; this representation is 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.

3.2 Constructing a grid in the event-driven approach

Figure 2 provides the general algorithm used to construct a grid for the event-driven model. The algorithm requires as input a data structure that maintains information about the streets. While iterating through the number of simulated lights on the grid, streets corresponding to each direction are visited. If a direction has

not been previously marked as visited, then inbound and outbound service queues are created for streets in that direction. That direction's construction is then labeled as COM_LINK. A COM_LINK in the model is a place in the network where inter-processor communication is required in the parallel implementation. If another grid lies logically adjacent to that direction, then the construction is labeled as SOURCE_SINK. A source is a place in the traffic network where a car can enter the system; this activity corresponds to a car entering a street where the street is on the boundary of the network. Similarly, a sink is a place in the traffic network where a car can exit the system. All of the boundary streets illustrated in Figure 1 are sources and sinks. If a direction is visited a second time the construction is changed from SOURCE_SINK to PASS_WITHIN to signify that the streets in that direction connect two nodes local to this processor. The algorithm to construct the grids for the time-driven approach is different (see [5]), however, the net effect is exactly the same.

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 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 [7] 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.

3.3 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

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

begin SimulateTraffic
   LocalTime = 0
    while LocalTime <= MaxSimTime loop
          while there are more null messages in the receive buffer loop
                process the null message
          end while
          while there are more car messages in the receive buffer loop
                process the car message
          end while
          if you can proceed without violating the 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 4: General algorithm for parallel simulation of a traffic flow network using the time-driven approach.

between the two approaches.

Figure 4 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 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 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 5. The algorithm uses an event_list, a main loop, and five distinct event-types. The main loop iterates until the local time is greater

```
algorithm    SimulateTraffic
input        Futureevents
output       simulation time
init         travel_time to 10 , crossing_time to 2,
             and next_light to 5
events       ARRIVE, LEAVE, and LIGHT

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

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 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 every tick. Unproductive checking would be expected for sparse 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 $O(logn)$ 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.

Statistic	*cars generated*	*cars exiting*	*total messages*	*null messages*	*car messages*	*cars/ message*
Event	3,451,303	3,438,014	479,952	1,665	478,287	21.3
Time	3,359,579	3,345,447	479,921	534	479,387	20.8

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.

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*, [3] the probability that a car is generated at a source, and the number of lights (and streets) in the model. 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 varies from 16 to 576 lights in the traffic network. Increasing the number of lights induces a corresponding increase in the number of streets and, resulting 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*.

[3] 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.

4.1 Summary statistics

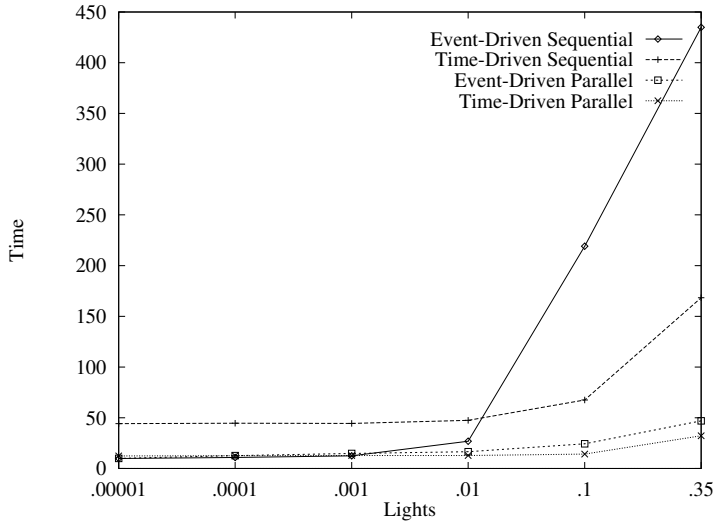
The summary statistics in Table 1 illustrate that the implementations of the event-driven and time-driven models represent the same traffic network. The summary was garnered by executing each of the parallel implementations on 16 Sun SLC workstations. The data in Table 1 represents averages over 30 executions 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 thirty-six 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. Column four of Table 1 indicates that the number of null messages required by the event-driven approach is 1,665 and the number of null messages required by the time-driven approach is 534. The number of car messages for the event-driven approach is 478,287 and for the time-driven approach is 479,387.

The last column in Table 1, *cars/message*, indicates that an average of 21.3 and 20.8 cars were packed into a message for the event-driven and time-driven approaches respectively. This is a reasonable average for messages since both models contained an average of 3.5 cars per street and there are 6 streets on the boundary of each grid when the 576 lights are partitioned across 16 processors. Thus, each communication contains an average of 21 cars, or 3.5*6 cars.



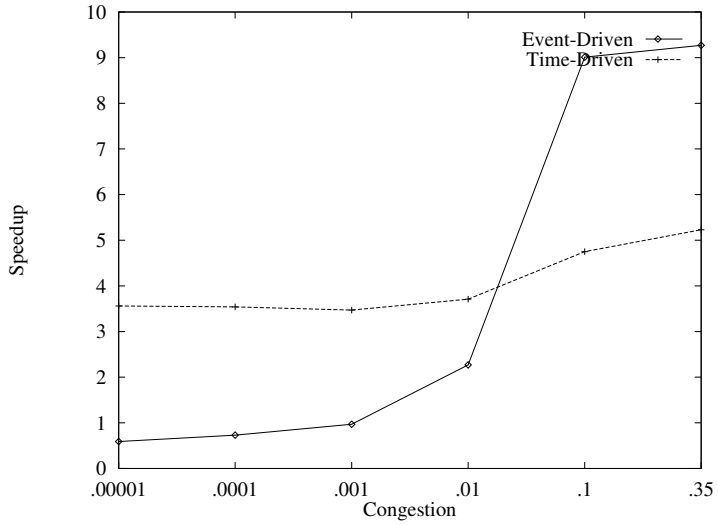
	Approach	Traffic Flow					
		100	*1,000*	*10,000*	*100,000*	*1M*	*3.5M*
Seq	Event	9.8	10.8	12.5	26.9	219.1	434.9
	Time	36.3	37.4	37.6	43.4	83.9	165.8
Par	Event	9.9	12.7	14.8	16.5	24.3	46.9
	Time	11.3	11.5	11.2	12.2	14.0	17.6

Figure 6: 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 16 SLC Sun workstations.

4.2 The effect of increased *traffic flow* on the two models

Figure 6 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. The table below the graph in Figure 6 contains the time in minutes for the various *traffic flow* densities for each of the models.

For the graph of Figure 6, the solid line (diamond) and the dashed line (plus sign) compare the sequential execution times of the models; these lines illustrate the behavior of the two models for sparse and dense *traffic flow* and they correlate with the results of previous research [12]. 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



traffic flow	*100*	*1,000*	*10,000*	*100,000*	*1M*	*3.5M*
Event	.59	.73	.97	2.27	9.01	9.27
Time	3.21	3.25	3.36	3.56	5.99	9.12

Figure 7: Comparison of speedup for varying densities of *traffic flow* in the network.

to 3.5 million, the time-driven implementation executed faster than the event-driven implementation.

The *break-even* point in the comparison of the sequential executions is approximately 100,000 cars, where the *break-even* point indicates the *traffic flow* density that results in an equivalent number of events/activities in the event-driven and the time-driven implementations. Since the implementations executed for 100,000 clock ticks and 100,000 cars are generated in the simulation, we have an average of one car being generated at each tick. Thus, the *break-even* point occurs when the event-driven model must process one event each clock tick and the time-driven model must process one event each clock 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; however, the time-driven implementation must scan a list of 2304 events (576 lights x 4 service queues at each light) on every clock tick, regardless of the time of the next event. For *traffic flow* densities greater than 100,000 cars, the time-driven implementation has many events to process at each clock tick where each event entails an O(1) queue operation; however, the event-driven implementation has many events to process at each clock tick where each event entails an O(log n) priority queue operation.[4] Note that for the sparsest *traffic flow* of 100 cars, the

[4]future work includes an event-driven implementation that uses the O(1) priority queue of [2]

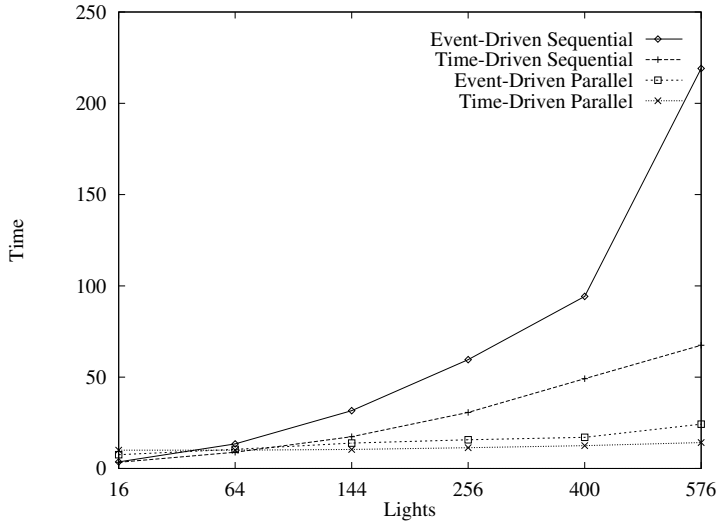
event-driven implementation ran 3.7 times faster than the time-driven implementation and for the densest *traffic flow* of 3.5 million cars, the time-driven implementation ran 2.6 times faster than the event-driven implementation.

Furthermore, the ratio of the the sequential execution times of the two models and thus, the two lines in the graph of Figure 6, is $logM$ where M is the number of events[5] in the system. To see this, consider the case where there is an average of one car generated on each clock tick and that on each clock tick the time-driven implementation scans a list of activities, of constant size, and performs an $O(1)$ queue operation. Similarly, the event-driven implementation processes an event on each clock tick, performing an $O(logM)$ priority queue operation. Thus, the ratio is $logM$.

A similar pattern of execution times is illustrated in Figure 6 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 **execution time** is reversed when considering the **speedup** achieved in the parallel executions of the two models. Figure 7 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 7 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.

[5] the number of events in the system is proportional to the number of cars in the system.



	Approach	Number of Lights					
		16	*64*	*144*	*256*	*400*	*576*
Seq	Event	3.61	13.44	31.67	59.59	94.24	219.1
	Time	4.7	13.4	26.5	43.9	68.3	83.9
Par	Event	7.57	10.46	13.93	15.7	17.08	24.3
	Time	8.9	9.9	10.2	11.9	12.0	14.2

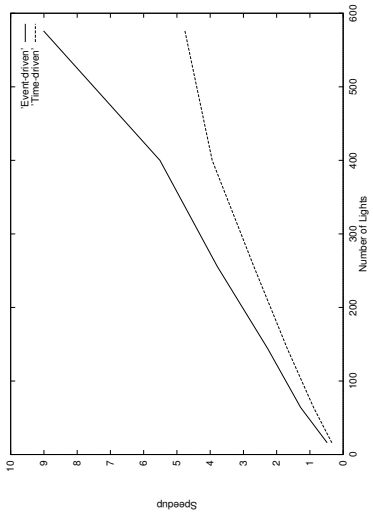
Figure 8: The graph in this figure illustrates the impact of workload on the average running time of the simulation. The solid line in the graph shows the increase in running time as the workload increases for the parallel execution of the event-driven model. The dashed line in the graph shows the increase in running time as the workload increases for the parallel execution of the time-driven model. The table includes the numerical data for both the sequential and parallel executions of both models.

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 8 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 8 and Figure 6 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 9, illustrates that increasing the *workload*, improves speedup. Previous research has shown that



Intersections	*16*	*64*	*144*	*256*	*400*	*576*
Event	0.48	1.28	2.27	3.79	5.51	9.01
Time	0.50	1.35	2.59	3.65	5.69	5.90

Figure 9: Graphical comparison of speedup for varying numbers of lights in the system. All executions used 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.

when communication is expensive, increasing the amount of work performed by each processor can offset the communication cost and result in improved speedup [3, 4, 5]. The graph in Figure 9 illustrates that the same increase in speedup was achieved in both models.

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 [12].

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).

6 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

[1] D. Baezner, C. Rohs, and H. Jones. U. S. army modsim on jade's time warp. *Proceedings of the 1992 Winter Simulation Conference*, pages 665–671, December 1992.

[2] R. Brown. Calendar queues: A fast O(1) priority queue implementation for the simulation event set problem. *Communications of the ACM*, 31(10):1220–1227, October 1988.

[3] C. D. Carothers, R. M. Fujimoto, Y. B. Lin, and P. England. Distributed simulation of large scale PCS networks. *Proceedings of the Second International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 2–6, January 1994.

[4] Phillip M. Dickens, Philip Heidelberger, and David M. Nicol. A distributed memory LAPSE: Parallel simulation of message-passing programs. *Proceedings of the Second International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 32–38, January 1994.

[5] J. T. Douglass and B. A. Malloy. Using a shot clock to design an efficient parallel distributed simulation. *Proceedings of the 1994 Winter Simulation Conference*, pages 1362–1369, December 1994.

[6] R. M. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):31–53, October 1990.

[7] A. P. Galluscio, J. T. Douglass, B. A. Malloy, and A. J. Turner. A comparison of two methods for advancing time in parallel discrete event simulation. Technical Report 95-105, Clemson University, May 1995.

[8] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. PVM 3 user's guide and reference manual. *Oak Ridge National Laboratory*, ORNL/TM-12-87:108, May 1993.

[9] D. R. Jefferson. Virtual time. *Transactions on Programming Languages and Systems*, pages 404–425, July 1985.

[10] V. Madisetti and D. Hardaker. Synchronization mechanisms for distributed event-driven computation. *ACM Transactions on Modeling and Computer Simulation*, pages 12–51, January 1992.

[11] V. Madisetti, J. Walrand, and D. Messerschmitt. Wolf: A rollback algorithmn for optimistic distributed simulation systems. *Proceedings of the 1988 Winter Simulation Conference*, pages 296–305, December 1988.

[12] R. E. Nance. On time flow mechanisms for discrete system simulation. *Management Science*, 18(1):59–73, September 1971.

[13] R. Righter and J. C. Walrand. Distributed simulation of discrete event systems. *Proceedings of the IEEE*, 77(1):99–113, January 1989.

[14] B. W. Unger, J. Cleary, A. Dewar, and Z. Xiao. A multi-lingual optimistic distributed simulator. *Transactions of the Society for Computer Simulation*, pages 121–152, June 1990.