Quantitative Driven Optimization of a Time Warp Kernel

Sounak Gupta
Dept of EECS, PO Box 210030
Cincinnati, OH 45110–0030
sounak.besu@gmail.com

Philip A. Wilsey
Dept of EECS, PO Box 210030
Cincinnati, OH 45110–0030
wilseypa@gmail.com

ABSTRACT

The set of events available for execution in a Parallel Discrete Event Simulation (PDES) are known as the pending event set. In a Time Warp synchronized simulation engine, these pending events are scheduled for execution in an aggressive manner that does not strictly enforce the causal relations between events. One of the key principles of Time Warp is that this relaxed causality will result in the processing of events in a manner that implicitly satisfies their causal order without paying the overhead costs of a strict enforcement of their causal order. On a shared memory platform the event scheduler generally attempts to schedule all available events in their Least TimeStamp First (LTSF) order to facilitate event processing in their causal order. By following an LTSF scheduling policy, a Time Warp scheduler can generally process events so that: (i) the critical path of the event timestamps is scheduled as early as possible, and (ii) causal violations occur infrequently. While this works effectively to minimize rollback (triggered by causal violations), as the number of parallel threads increases, the contention to the shared data structures holding the pending events can have significant negative impacts on overall event processing throughput.

This work examines the application of profile data taken from Discrete-Event Simulation (DES) models to drive the simulation kernel optimization process. In particular, we take profile data about events in the schedule pool from three DES models to derive alternate scheduling possibilities in a Time Warp simulation kernel. Profile data from the studied DES models suggests that in many cases each Logical Process (LP) in a simulation will have multiple events that can be dequeued and executed as a set. In this work, we review the profile data and implement group event scheduling strategies based on this profile data. Experimental results show that event group scheduling can help alleviate contention and improve performance. However, the size of the event groups matters, small groupings can improve performance, larger groupings can trigger more frequent causal violations and actually slow the parallel simulation.

SIGSIM-PADS'17 May 24-26, 2017, Singapore, Singapore

DOI: http://dx.doi.org/10.1145/3064911.3064932

Keywords

Pending event set; Profile Guided Optimization; Event Scheduling; Lock contention; Parallel and distributed simulation; Time warp

1. INTRODUCTION

The set of events available for execution in a Parallel Discrete Event Simulation (PDES) are known as the *pending event set*. In an Time Warp synchronized simulation engine, these pending events are scheduled for execution in an aggressive manner that does not strictly enforce the causal relations between events [6, 10]. One of the key principles of Time Warp is that this relaxed causality will result in the processing of events in a manner that implicitly satisfies their causal order without paying the overhead costs of a strict enforcement of their causal order. Unfortunately, the event processing granularities of most discrete event simulation models are generally quite small which aggravates contention to the pending event shared data structures of a parallel simulation engine on a multi-processor platform. To alleviate this, some researchers have attempted to exploit lock-free methods [7], hardware transactional memory [8, 16], or synchronization friendly data structures [5]. Unfortunately, lock-free methods have additional overheads when incorporating sorting and deletion operations. While hardware transactional memory does reduce overhead this is mostly due to higher performing locking methods rather than providing any significant overall reduction in contention. Restructured data structures can be helpful to reduce contention, but even then we have found that contention remains an issue that needs to be addressed.

The work in this paper attempts to draw on the success of the computer architecture community in the use of application profile data to optimize the design and implementation of a compute platform [9]. In particular, we examine profile data from discrete event simulation models to help guide the design and optimization of scheduling strategies for the pending event set in a Time Warp synchronized simulation kernel. In this study, we focus on a single node shared memory platform to demonstrate the application of quantitative based design optimization. The profile data we use comes from previous work to build a system to profile discrete event simulations [20]. In this work, we will focus specifically on the data regarding *event chains* as reported in [20]. From that work, the event chain data data illustrates how many events could potentially be dequeued and

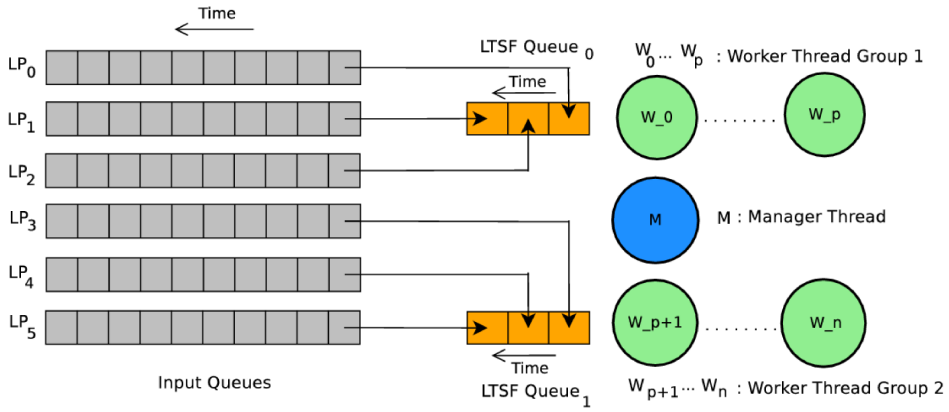

Figure 1: Pending Event Set in warped2

scheduled together such that they could be guaranteed ready to be executed. The data from that paper suggests that for some simulation models, a Time Warp simulator could dequeue multiple events with each event scheduling activity and successfully commit those events without triggering many rollbacks. In this paper, report on our experiences to leverage this event chain data into two different possible event scheduling strategies.

The remainder of this paper is organized as follows. Section 2 reviews some of the work related to scheduling events in parallel simulation. Section 3 reviews the previous work that quantifies the event organization in several discrete event simulation models. The data from this quantification motivations the approaches for scheduling events from the pending event set studied in this paper. Section 4 reviews the organization of the pending events in the WARPED2 parallel simulation engine which is the platform that our experimental analysis is performed. Section 5 describes the different methods that are explored for organizing, managing, and optimizing the pending event set to support LTSF event scheduling. Section 6 presents a performance results comparison between the different scheduling strategies for the pending event set. Finally, Section 7 contains some concluding remarks.

2. RELATED WORK

The Georgia Tech Time Warp (GTW) [4], a parallel discrete event simulator, is an implementation of the Time Warp mechanism proposed by Jefferson [10]. It was designed for discrete-event simulation applications having relatively small event granularity that could be executed on cache-coherent, shared-memory multiprocessors. GTW system consists of a group of logical processes (LPs) that communicate among themselves by exchanging time-stamped events (messages). The execution of each LP is message driven. Positive messages sent between LPs are directly inserted into the *Message Queue* while the negative (or anti) messages are similarly inserted into the *Cancel Queue*. Frequent access to message and cancel queues can lead to drastic increase in contention. To alleviate this problem, GTW designers introduced the concept of batch processing several unprocessed events within a specified batch interval. Several events would be processed in a batch by temporarily ignoring any rollbacks or cancellations. Though this approach would have reduced contention in the message and cancel queues, it can lead to the problem of overly optimistic execution in Time Warp. This refers to the situation where some LPs may progress too far ahead of others thereby leading to inefficient memory use and/or longer rollbacks. The authors, predicting this, implemented a 3-fold mechanism to tackle this problem. Blocking I/O events prevent LPs from progressing beyond a stated simulation time. Thus, an over-optimistic LP could be blocked from progressing by introducing a "dummy" blocking I/O event. The second mechanism employs the coarser approach of *simulated time barrier*. The idea borrows heavily from Bounded Time Warp synchronization protocol [17] which is a window-based simulation mechanism. The third mechanism deals with the frequency of fossil collection, which is a way to reclaim memory from events and states that are no longer being used.

WARPED [12, 14], built at University of Cincinnati, is a faithful implementation of Jefferson's classic model. Here LPs have their own input, output, and state queues. The initial kernel was designed for a distributed compute platform and the only parallelism available was among processes communicating by message passing. With the advent of newer generation of multicore processors, WARPED was re-designed (and renamed to WARPED2) such that is supported both fine-grained parallel threaded execution (pthreads) as well as process based parallelism that uses message passing. The pending event set in WARPED2 follows the design suggested by Ronngren *et al* [15] (and also used in several other kernels). There are two event pools, namely `Unprocessed` and `Processed` events. Events that are yet to be executed are stored in the `Unprocessed` pool. `Processed` pool stores processed events till the next fossil collection (*i.e.,* until the next GVT calculation). Adhering to the Time Warp mechanism, WARPED2 greedily processes events without strict adherence to their causal order. The simulation rolls back to the last known consistent state on detecting causal violations in any LP.

WARPED2 uses a two-level hierarchical organization for the *Pending event set* (Figure 1). On the first level, unprocessed events for each LP are stored in separate timestamp-sorted *Input Queue*. The second level is the timestamp-sorted *Schedule Queue* (often referred to as the Least Timestamp First or LTSF queue). The smallest unprocessed event from each LP is enqueued into this data structure. Dickman *et al* [5] proposed the use of multiple *LTSF Queues* to reduce

Algorithm 1: Event execution loop schematics

```
1  Lock the LTSF Queue linked to the worker thread;
2  Dequeue the smallest event e_m from that LTSF Queue;
3  Unlock that LTSF Queue;
4  while e_m is a valid event do
5      Process e_m (assume e_m belongs to LP_i);
6      Lock Input Queue for LP_i;
7      Move e_m from Input Queue to the Processed Queue;
8      Read the next smallest event e_n from Input Queue;
9      Lock the LTSF Queue linked to the worker thread;
10     Insert e_n into LTSF Queue;
11     Dequeue the smallest event e_m from the LTSF
       Queue;
12     Unlock the LTSF Queue;
13     Unlock Input Queue;
14 end
```

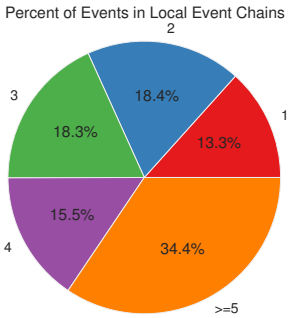

Figure 2: Percent of Events in the Event Chains in an Automobile Traffic Model

contention on a many-core processing platform. The main event processing loop in the WARPED2 simulation kernel is depicted in Algorithm 1. In WARPED2, each event processing thread is called a *worker thread* that continuously executes events from the pending event set until some termination condition is satisfied. A separate *manager thread* processes remote communication and Time Warp housekeeping functions, including termination detection.

3. MOTIVATING RESULTS FROM QUANTITATIVE STUDIES

Contention for access to the pending event set is a problem that we have studied for several years in the WARPED2 simulation kernel. We have explored various approaches such as: different organizations of data structures [5], model partitioning [1], lock-free algorithms for the pending event set[7], and transactional memory to access the pending event set [8]. While each provides a measure of relief for the contention, each have either limited success or disadvantages that inhibit their widespread use.

In a related study, we have pursued the development of methods to quantify the runtime profile of events in a discrete event simulation model [20]. This study developed tools to analyze the runtime traces of events processed by a simulation engine in order to discover certain properties about the events generated and processed by the simulation model. While the work profiled various characteristics such as events available for execution and the communication density of objects in the simulation, one characteristic that was captured called *event chains* suggested an optimization to the pending event set in a parallel simulation engine.

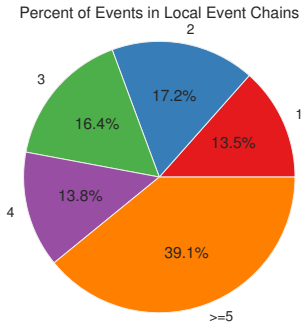

Figure 3: Percent of Events in the Event Chains in a Portable Cellular Service Model

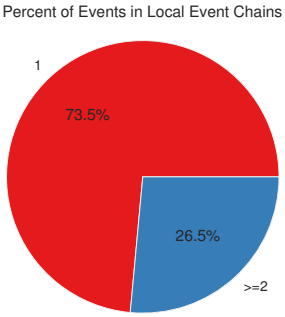

Figure 4: Percent of Events in the Event Chains in a Model of Disease Propagation through a Population

In particular, event chains are the collection of events from the pending event set of an LP that could potentially be executed as a group. That is, at a specific simulation time, an event chain would contain all of the events that time would be available in the pending event set for immediate execution at that time. Thus, we independently examine the pending event set of each LP. Beginning at time zero, the a chain of events is constructed and its maximum length counted. All of the events in that chain are treated as one and the algorithm then advances to the next event following the last in the chain to determine the length of the next chain. While the previous paper [20] classifies chains into three types (*local*, *linked* and *global*), in this paper we consider only local chains.

Our hypothesis is that if event chains of length greater than one are common, an interesting optimization to a simulation kernel might be to dequeue multiple events with each execution thread to execute as a block. Using the data from the previous study [20], Figures 2, 3, and 4 show the percentages of total events within each chain class. This data shows that events in two two of the simulation models (Figures 2 and 3) are in chains of length ≥ 2 and that they should therefore be candidates for scheduling in groups. However, the third simulation model (Figure 4) has nearly 74% of events in chains of length 1.[1]

[1]The pie chart of Figure 4 aggregates data for chains of length greater than two are nearly zero. Because the plotting

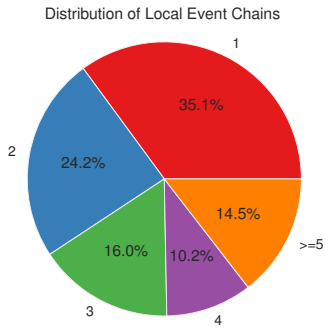

Figure 5: Event Chains in an Automobile Traffic Model

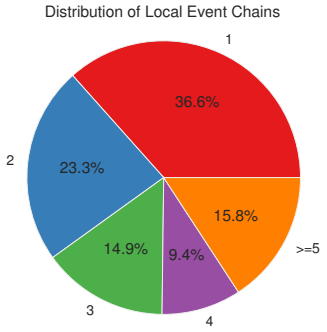

Figure 6: Event Chains in a Portable Cellular Service Model

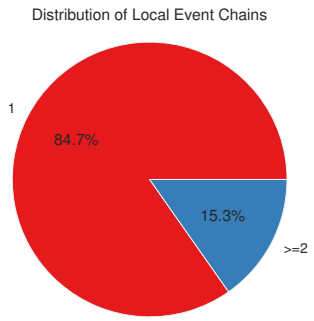

Figure 7: Event Chains in a Model of Disease Propagation through a Population

An alternate view of the event chain data is to show the number of chains of various lengths that occur throughout the simulation. This will help to demonstrate the percentages of multi-event scheduling opportunities that exist in the simulation. This organization of the data is shown in Figures 5, 6, and 7. The data in these pie charts highlight the percentage of chains of length 1 to 4 and greater than or equal to 5 that were found in each simulation model. Examining the first two simulation models (Figures 5 and 6), we can see that if two events are dequeued for each event scheduling activity, the simulator should find two immediately committable events more than 64% of the time; if three events are dequeued per scheduling activity then the simulator should find three immediately committable events approximately 40% of the time. As before, the third simulation model (Figure 7) shows that the simulator would only see two committable events approximately 15% of the time and nearly zero opportunities for finding chains of committable events greater than length two.

The profile data suggests that some simulation models will benefit from an event scheduler that distributes more than one event at a time from the pending event set. However, this analysis is from a conservative viewpoint of the event's availability. In part, optimistic synchronization benefits when causal relations are not strictly limited by a global time order in the simulation. Therefore, it could very well be that experimentation will show benefits from even larger counts of events being scheduled as a group than the profile data suggests.

4. PENDING EVENTS IN WARPED2

The WARPED2 Time Warp simulation kernel [19] is designed for execution on a single SMP processor or on a Beowulf cluster of SMP processors communicating with each other using MPI. WARPED2 uses a two-level hierarchical data structure to maintain the pending event set, namely: (i) a sorted list of pending events for each Logical Process (LP), and (ii) a set of one or more more scheduling pools of events called the LTSF queues. Each LP contains its list of pending events. This is a shared data structure with a unique lock assigned to each LP. The lowest timestamped event from each LP is placed into one of the LTSF queues. Each LTSF is also assigned a unique lock. Figure 1 presents an overview of this hierarchical design. A static partitioning of the LPs to the different nodes of the cluster is achieved using a profile guided partitioning process that optimizes a min-cut of the number of events exchanged between LPs [1]. This profile data is also used to assign LPs within a node to one of the LTSF queues. If there is only one LTSF queue, all LPs are assigned to that LTSF queue.

The number of LTSF queues on each compute node of the simulation is a runtime configuration parameter. The collection of threads that execute events (called worker threads) on each node are then statically assigned to a specific LTSF queue on that node. The worker threads process events from its assigned LTSF queue and inserts any generated events di-

software overwrites the labels making the graphic difficult to visualize, all chains longer than two are shown together. Likewise Figure 7 (discussed in the next paragraph) also aggregates the chain data for all chains of length two and above. For all practical purposes, the measurable number of chains in the epidemic model are limited to sizes of one or two events.

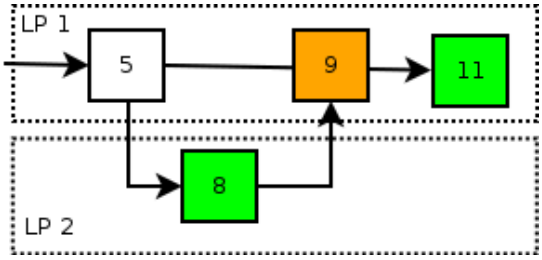

Figure 8: Event Causality

rectly into the corresponding LP event queue (events generated for remote nodes are placed into a message send queue). If a newly generated event defines a new lowest event for the LP, the worker thread also replaces the entry in the LTSF queue containing events for that LP. Finally there exists a manager thread on each node that performs housekeeping functions such as GVT and termination management [6]. The manager thread also performs the remote communication (sending and receiving) of events with other nodes. Thus, the manager thread must also sometimes access the shared data structures of the pending event set.

The hierarchical structure of the pending event set in WARPED2 provides for a highly effective scheduling of events that leads to infrequent rollbacks for many simulation models [5, 19]. On a single multi-core compute node using one LTSF queue, the WARPED2 simulator will generally experience zero rollbacks. However, as the number of worker threads increases beyond 4-6, contention for the LTSF queue diminishes overall performance as additional threads are added. In these situations, a multi-LTSF queue configuration can regain scalability [5]. However, with more than one LTSF queue, the scheduling of events on each node to follow the critical path of timestamped event execution becomes more problematic and rollbacks can increase. In this paper, we examine structuring the worker threads to dequeue multiple events per access to the shared pending event list data structures. This should further reduce contention and provide improved scalability with a fewer number of LTSF queues. The strategies explored and the results of experiments with them are described in the next two sections.

5. GROUP SCHEDULING: BLOCKS AND EVENT CHAINS

Group Scheduling is a opportunistic approach to scheduling pending events. Processing events in groups helps to reduce the frequency of access to the key shared data structure in the pending event set and therefore should help reduce contention contention to this critical resource. That price is increased risk of causal violation. Figure 8 contains an example illustrating a causal chain. Certain events in the pending event pool have a chronological order and cannot be processed greedily out of order. If such events are processed out of order, it leads to a *Causal Violation*. The benefits gained from scheduling schemes such as *Group Scheduling* rests on the rationale that time saved from reduced contention exceeds time wasted on rollbacks due to increased causal violations.

In this paper, we explore two different approaches to scheduling groups of events from the pending event set. In particular we consider scheduling groups of events from the

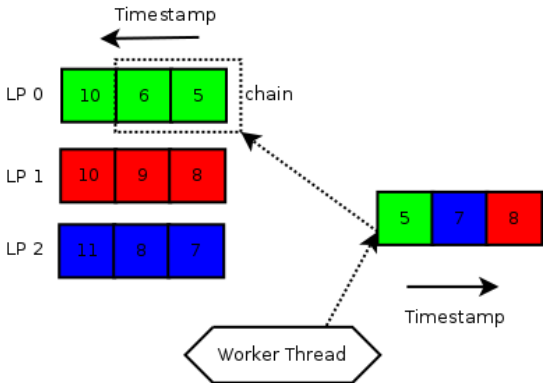

Figure 9: Chain Scheduling

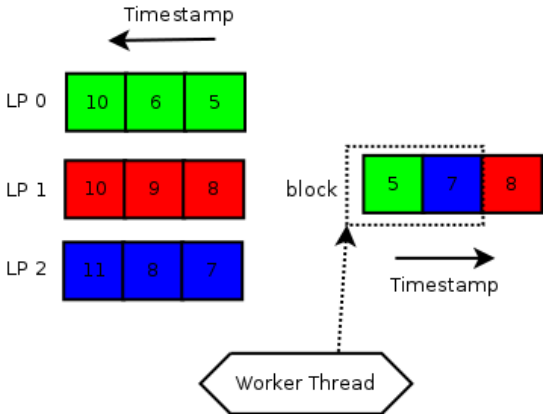

Figure 10: Block Scheduling

LPs as outlined in the profile study reported in [20]. We call this *chain scheduling*. Based on our initial success with chain scheduling, we restructured the solution to simply pull groups of events from the `LTSF queue`. We call this *block scheduling*. Each is described more fully in the next subsections.

5.1 Chain Scheduling

Figure 9 shows the schematics of *Chain Scheduling*. In this scenario, the smallest event from a `LTSF Queue` is considered. A group of consecutive events from the *Logical Process* linked to that smallest event forms the *chain*. The size of this chain (also referred to as *chain size*) is a configurable parameter. All events from this *chain* are processed in the same execution cycle. Output events, generated due to processing of these events, are either sent immediately or stored and sent in bulk but with a delay. Both these output event sending schemes have been studied in Section 6.2.

5.2 Block Scheduling

Figure 10 shows the schematics of *Block Scheduling*. In WARPED2, the smallest events from each *Logical Process* are placed in a timestamp-ordered priority queue (generally referred to as `LTSF` pool). Instead of pulling out one event per processing cycle (as is the usual scenario) from the `LTSF` pool, each worker thread dequeues a group of consecutively ordered events to form an event *block*. The size of this block (also referred to as *block size*) is a configurable parameter.

All events in a *block* are processed in the same execution cycle. The output events, generated as a result of this block processing, are sent out immediately after being created in order to minimize the effect of causal violations.

While the current version of block scheduling dequeues the next N consecutive events, we have considered dequeuing N at some unit distance forward in the LTSF queue (*e.g.,* every other event, every 3^{rd} event, and so on) in order to better distribute the lowest events in the `LTSF queue` throughout the various worker threads. The distance between events dequeued is referred to as *skip distance*.

6. EXPERIMENTAL RESULTS

In the experimental analysis of group scheduling, we use the three models examined in Section 3 of this paper. The next section describes these models and then results from the block and chain scheduling studies with these models are reported. Furthermore, the distribution of output events with group scheduling is treated in two different ways. In particular, in the first method (called the *delayed* method), output events were held until the entire group of events were executed and then the output events were distributed. In the second method (called the *immediate* method), output events were distributed immediately upon their generation from event processing. Overall the immediate method consistently outperformed the delayed results.

6.1 Simulation Models

This section describes the three WARPED2 simulation models used as benchmarks to study the performance implications of the changes discussed in Section 5. Both these models are based on similar simulation models used by other researchers in parallel simulation.

6.1.1 Traffic

This model simulates the flow of cars through a grid of intersections. It models how cars move through and between the intersections. The intersections in our model are four way with three lanes in each direction. This model has been adapted for WARPED2 simulator from the ROSS [3] simulator.

The intersections here are modeled as Logical Processes and cars moving through and between intersections have been modeled as Events. As shown in Figure 11, there are three event phases at every intersection: `Arrival`, `Direction Select`, and `Departure`. The `Direction Select` event triggers the model to determine the direction a particular car should travel on next in order to reach its target intersection. The departure and direction select events are self-generated while the arrival event is received from an adjacent intersection.

Each car in this model is randomly assigned a specific intersection as destination. Cars are initially distributed uniformly across all intersections. A car, after arriving at an intersection, tries to go in the direction that would get it closer to its target. The flow of cars going through an intersection is regulated using thresholds to spread out the traffic and limit congestion on certain roads. If a car is denied permission to travel in the direction of its choice, it randomly chooses either to wait or take an alternate route. Events follow an exponential distribution and a new event is generated when an event is processed.

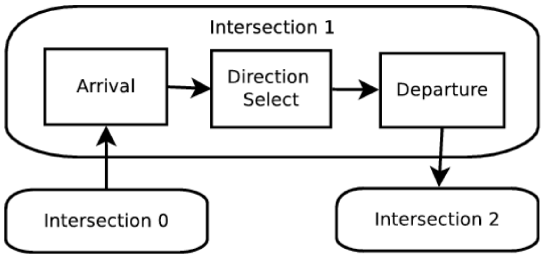

Figure 11: Sequence of events at each intersection

The State of a Logical Process keeps track of the status of traffic at each intersection. It consists of:

- the number of cars coming in from each direction,
- the number of cars going out in each direction,
- the total number of incoming cars at the intersection, and
- the total number of outgoing cars at the intersection.

To improve scalability of the traffic model, the grid has been modeled as a rectangular mesh (as is also done in ROSS). The grid size used for this simulation equals *1024 x 1024* which translates to a total of *1,048,576* LPs.

6.1.2 Portable Cellular Service (PCS)

This model simulates a wireless communication network that provides services for a number of subscribers (or *portables*). The service area of the network is divided into *cells*, each cell having a certain number channels that can be allocated for calls. A channel from the cell is allocated for an incoming or outgoing call. If no channels are available for allocation, the call is *blocked* [11]. Cells are the logical processes in this simulation and they are organized in the form of a rectangular grid as shown in Figure 12. The grid size used for this simulation equals *256 x 256* which translates to a total of *65,536* LPs.

Portables are allowed to move across adjacent cells with a wrap around occurring at the edges. The portables migrate to an adjacent cell after a time period that follows a Poisson distribution. The call is dropped if all the channels are busy on a cell to which the portable migrates to while on a call. This is called a *handoff block*. Call requests to a portable also follow a Poisson distribution. The State of a Logical Process keeps track of the channel count and call statistics at each cell. It consists of:

- Idle channel count,
- Call attempts count,
- Channel blocks count, and
- Handoff blocks count.

`NextCall`, `CallCompletion`, `PortableMoveOut`, and `PortableMoveIn` are the four event types in this simulation. `NextCall`, `CallCompletion`, and `PortableMoveOut` events are self-initiated by a cell whereas the `PortableMoveIn` event is sent to an adjacent cell with uniform randomness.

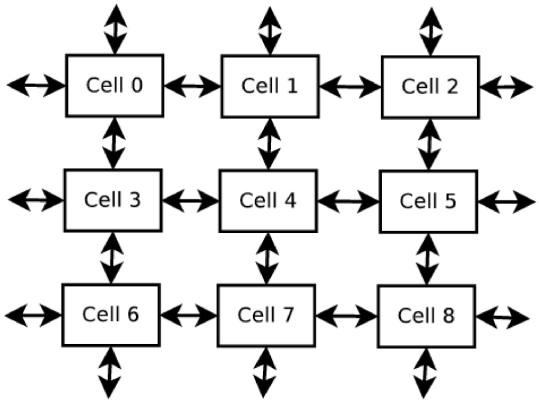

Figure 12: PCS Model Logical Processes

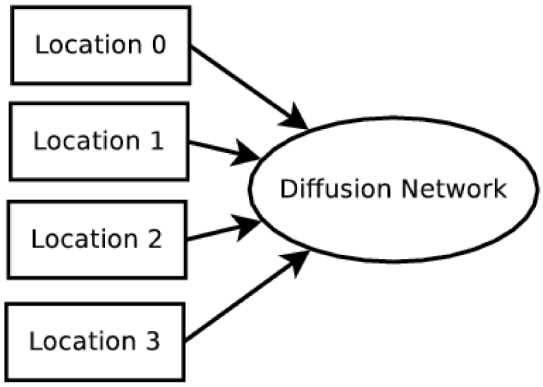

Figure 13: Epidemic Model

6.1.3 *Epidemic*

This models the spread of an infectious disease across a set of geographic locations, each housing a subset of the total population. It is based on a reaction-diffusion epidemic model proposed by Perumalla *et al* [13].

Each geographic location is represented in the simulation model as a logical process. Intra-location transmission of the disease between people is modeled as a probabilistic reaction function [2]. A finite state machine called *Probabilistic Timed Transition System (PTTS)* [2] models the spread of disease within an individual. Inter-location migration of population is modeled as a diffusion network based on the β model proposed by Watts *et al* [18]. Events in the epidemic model follow an exponential distribution. The number of logical processes used for this simulation model is *100,000*.

Figure 13 presents an overall illustration of this model.

6.2 Performance Data

These experiments were run on an Intel® Core™ i7-4770 (a quad core processor with two hardware threads per core) machine equipped with 32 GB of memory. All simulations were run using `STL MultiSet` as LTSF queue. A single LTSF Queue is used for all experiments discussed in this paper.

The performance metrics used in this study are:

- `Overall Speedup` for a particular configuration of worker thread and group size equals

$$\frac{\text{Max runtime among all configurations}}{\text{Runtime for that configuration}} \quad (1)$$

This metric is useful to compare the overall performance of a configuration and to study the combined effect worker thread count and group size has on group scheduling.

- `Relative Speedup` for a particular configuration of worker threads and group size equals

$$\frac{\text{Runtime for same thread count while using group size 1}}{\text{Runtime for that configuration}} \quad (2)$$

This metric is useful to study the effect of group size only on group scheduling by isolating the results from the effects of worker thread count.

- `Commitment Rate` for a particular configuration of worker threads and group size equals

$$\frac{\text{Total number of events processed for that configuration}}{\text{Total number of committed events}} \quad (3)$$

Total number of committed events is independent of effect of worker thread count and group size. This metric is useful to understand how the greedy aspect of group scheduling adversely impacts the simulation through causal violations.

6.2.1 *Traffic*

The model configuration mentioned in Subsection 6.1.1 was used for this series of experiments. The total count of events committed equals `14,881,335` in all cases.

Figure 14 contains the speedup results with the Traffic model using event scheduling. It shows a speedup of nearly 3 when groups of 2 and 4 events are scheduled using 8 threads and the trends with 4 and 2 threads mostly mirror 8 threads. Beyond a chain size of 4 events, however, performance begins to steadily degrade. The simulation runs slower for all threads when output events are sent out in `Delayed` mode compared to that in `Immediate` mode. Figure 15 shows similar relative speedup behavior until event chain size of 4 beyond which the relative performance degrades. Figure 16 plots the commitment rate for event chains and shows the steady increase in causal violations due to increase in size of event chains. The performance of event chain improves in spite of higher causal violations till a certain point. Beyond that event chain size, the benefit of lower contention in event chain management is offset by the increasing number of causal violations. It is worth noting that the commitment rate is nearly the same for different number of worker threads and output event sending modes. Though we have not yet analyzed this behavior thoroughly, our speculation is that the scope of causal conflict remains unchanged for the chain sizes considered for this experiment.

Figure 17 contains the speedup results with the Traffic Model using block scheduling. It shows that performance in case of 2 worker threads remains invariant when there is a change in event block size. It is to be noted that, due to large LP count, there are no causal violations for blocks of small

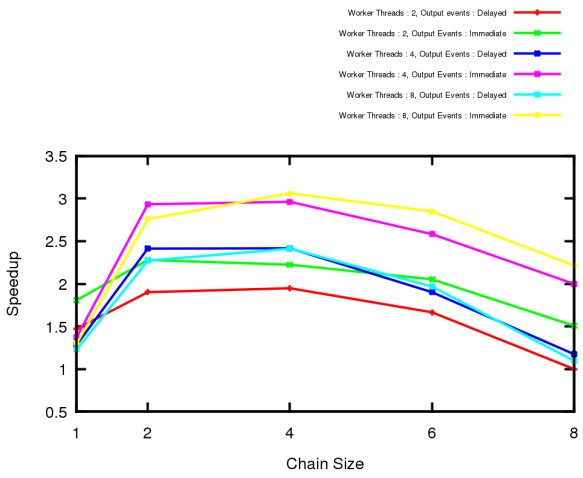

Figure 14: Overall Speedup of Traffic using Event Chains

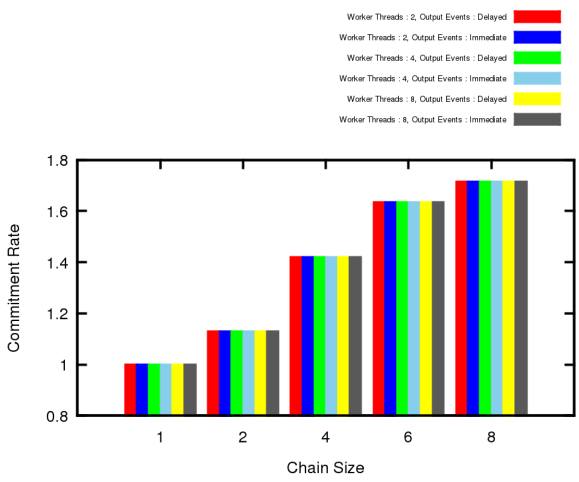

Figure 16: Commitment Rate of Traffic using Event Chains

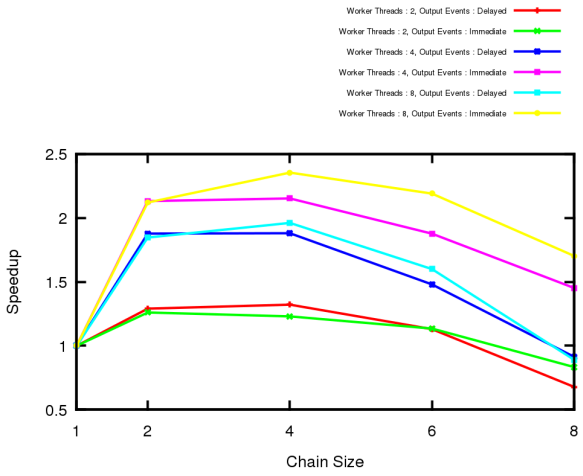

Figure 15: Relative Speedup of Traffic using Event Chains

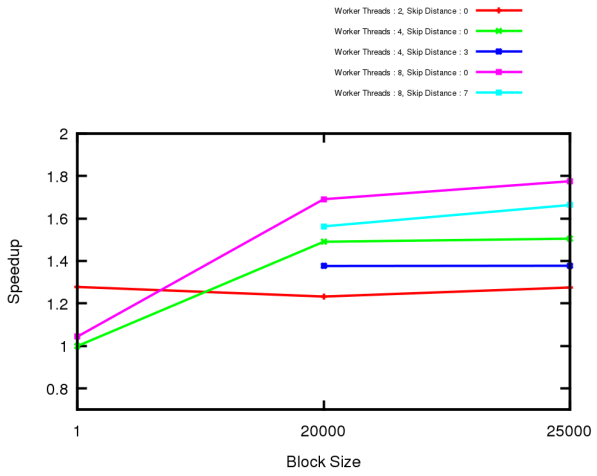

Figure 17: Overall Speedup of Traffic using Block Scheduling

sizes. Since one of the goals of this study is to study the impact of greedy processing of event blocks on event causality, the block size was increased to a point where causal violations could be observed. There is good overall speedup in case of 4 and 8 worker threads. However event blocks formed using `skip distance > 0` performs worse than event blocks formed using `skip distance = 0`. The same observations are true for relative speedup as shown in Figure 18. Figure 19 shows a relatively low number of causality violations when `skip distance = 0` and it increases when `skip distance > 0`.

6.2.2 PCS

The model configuration mentioned in Section 6.1.2 was used for this series of experiments. The total count of events committed equals `45,484,953` in all cases.

Figure 20 contains the speedup results with the PCS model when using chain scheduling. It shows decent good overall speedup for all threads. However, with the increase in event chain size, the performance steadily falls in case of 2 worker

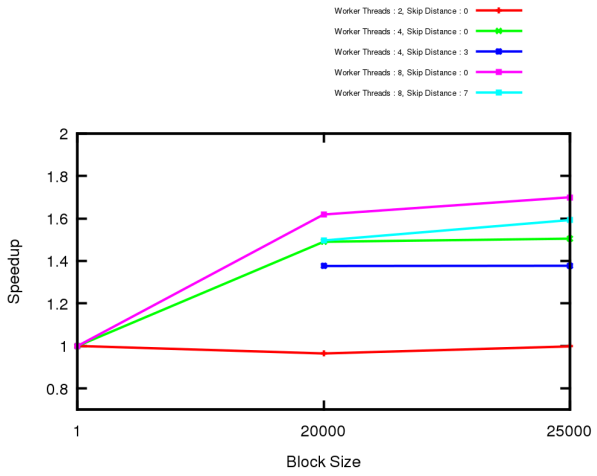

Figure 18: Relative Speedup of Traffic using Block Scheduling

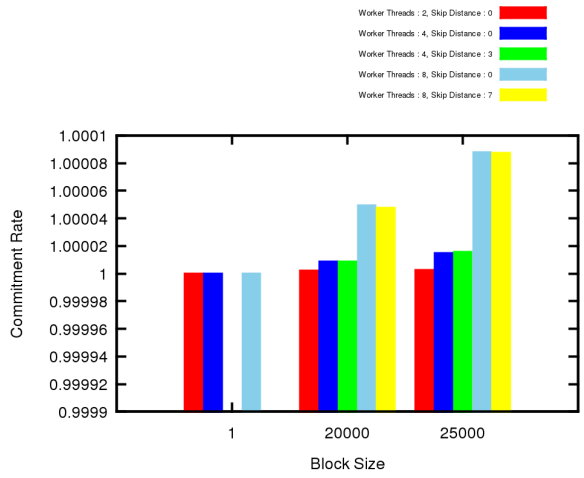

Figure 19: Commitment Rate of Traffic using Block Scheduling

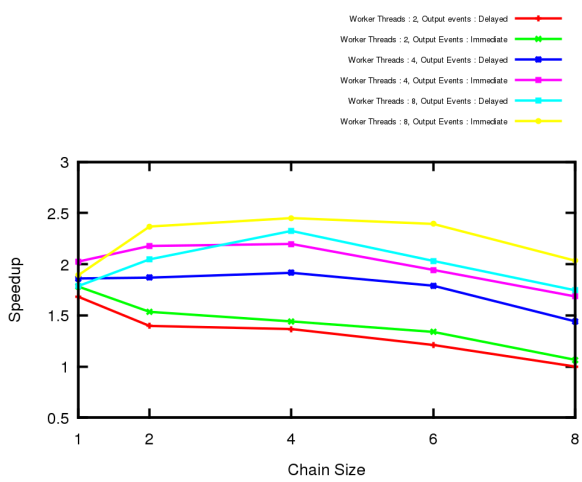

Figure 20: Overall Speedup of PCS using Event Chains

threads. This makes sense as contention should be minimal with only 2 worker threads. The performance remains fairly stable in case of 4 worker threads until the event chain size reaches 8. In case of 8 worker threads, the performance actually improves and peaks for event chain size of 4 before steadily degrading beyond that. Figure 21 shows that there is minimal or adverse relative speedup in case of 2 and 4 threads with increase in size of event chains. Only 8 worker threads shows improvement for event chain size of 4 before steadily degrading. Figure 22 plots the commitment rate for event chains and shows the steady increase in causal violations due to increase in size of event chains. The performance of event chain improves in spite of higher causal violations till a certain chain size. The benefit of lower contention in event chain management is offset by the increasing number of causal violations at that point. Similar to `Traffic` model, the commitment rate here is nearly the same for different number of worker threads and output event sending modes.

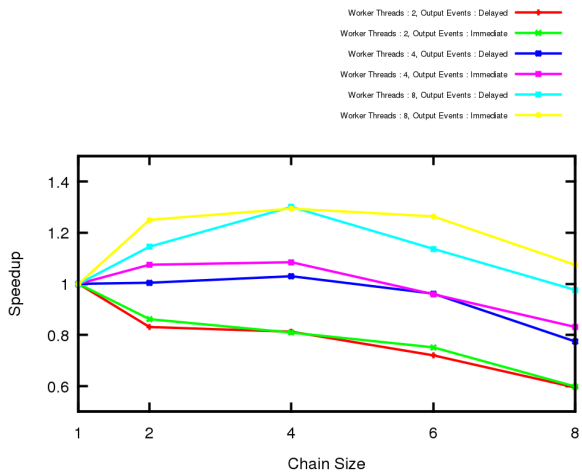

Figure 21: Relative Speedup of PCS using Event Chains

Figure 23 contains the speedup results with the PCS model when using block scheduling. Similar to the `Traffic` model, results presented are only for large block sizes in order to study the effect of greedy processing of event blocks on event causality. It shows that increase in event block size adversely affects overall performance in case of 2 worker threads. There is good overall speedup in case of 4 and 8 worker threads. However, event blocks formed using `skip distance > 0` performs worse than event blocks formed using `skip distance = 0`. The same observations are true for relative speedup as shown in Figure 24. Figure 25 shows relatively low number of causality violations when `skip distance = 0` and it increases marginally when `skip distance > 0`.

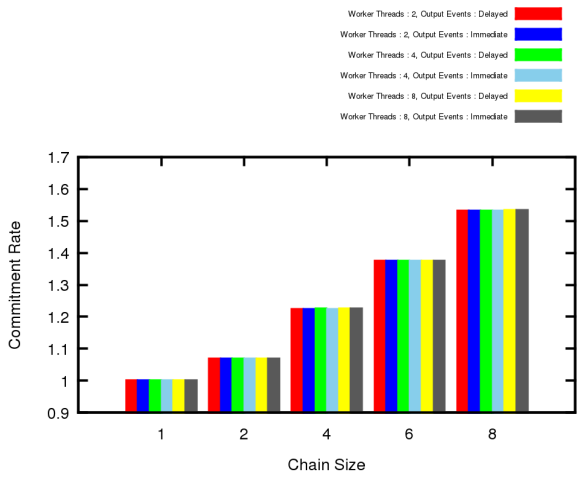

Figure 22: Commitment Rate of PCS using Event Chains

6.2.3 *Epidemic*

The model configuration mentioned in Subsection 6.1.3 was used for this series of experiments. The total count of events committed equals `30,676,881` in all cases.

If we review the profile results from Section 3, the evidence for the application of group scheduling in the Epidemic model is not encouraging. In fact, reviewing Figure 7

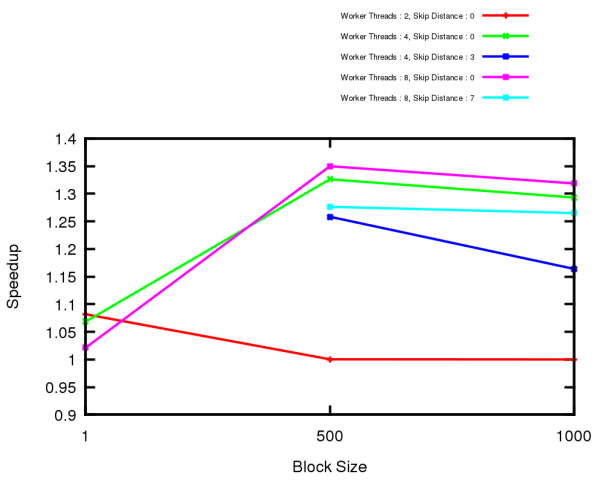

Figure 23: Overall Speedup of PCS using Block Scheduling

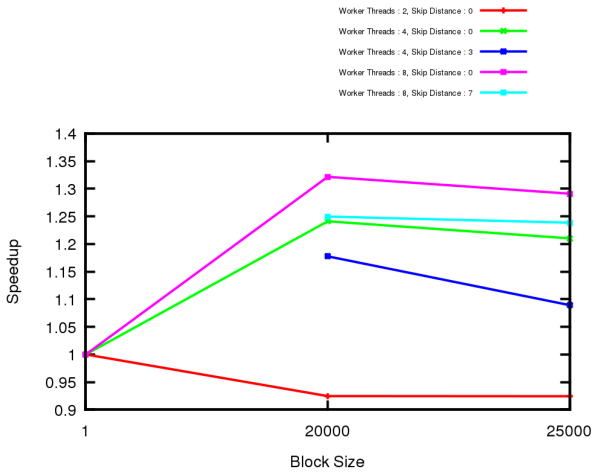

Figure 24: Relative Speedup of PCS using Block Scheduling

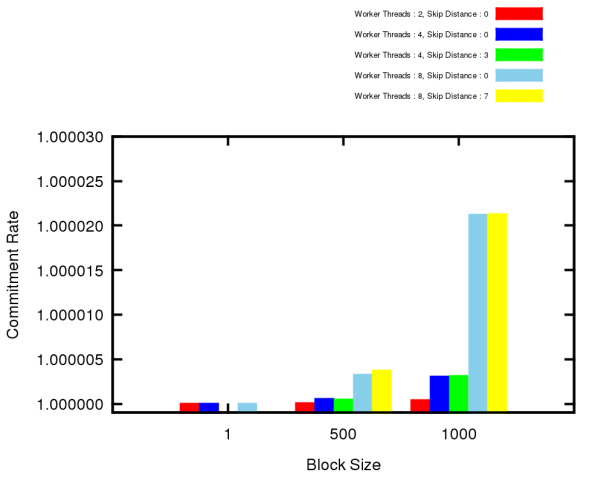

Figure 25: Commitment Rate of PCS using Block Scheduling

we can see that nearly 85% of the event chains in the Epidemic model are of length 1. As will be seen below, the experimental results mostly follow this result. While some modest improvements are seen, the results are not as dramatic as they are with the Traffic and PCS models shown above.

Figure 26 contains the speedup results with the Epidemic model using event scheduling. It shows minor speedup for all threads. The performance improves for event chain size of 2 in case of worker thread count of 4 and 8. Beyond this point performance degrades and remains stable. The simulation runs slower for all threads when output events are sent out in `Delayed` mode compared to that in `Immediate` mode. Figure 27 shows similar relative speedup behavior till event chain size of 2 beyond which the relative performance degrades and remains stable. Figure 28 plots the commitment rate for event chains and shows the increase in causal violations due to increase in size of event chains. The commitment rates in case of event chain sizes 4–8 are relatively similar. Since the overall and relative speedup values are also stable for event chain sizes 4–8, it indicates the benefit of using event chains to offset the contention problem when there are high number of worker threads. Similar to `Traffic` and `PCS` models, the commitment rate is nearly the same for different number of worker threads and output event sending modes. Though we have not yet analyzed this behavior thoroughly, we speculate that the scope for causal conflict remains unchanged when the chain size is considered for this experiment.

Figure 29 contains the speedup results with the Epidemic model using event scheduling. Similar to the `Traffic` and `PCS` models, results discussed are for large block sizes only since the focus of this study is the effect of greedy processing of event blocks on event causality. It shows that performance in case of 2 worker threads remains invariant when there is a change in event block size. There is much better speedup in case of 4 and 8 worker threads than was achieved with chain scheduling. The same observations are true for relative speedup as shown in Figure 30. Figure 31 shows no causality violations for worker thread count 2 and 4. Causality violations are observed in case of 8 worker threads. This indicates that causal chains do not occur frequently in the epidemic event stream.

7. CONCLUSION

The use of profile data from Discrete Event Simulation Models to develop pending event set scheduling strategies for a Time Warp synchronized simulation kernel is studied. The profile data suggested that two of the studied simulation models should benefit from the scheduling of multiple events during the event scheduling step of the simulation engine. Profile data from a third simulation model suggested that the opportunity to gain speedup from group scheduling would not be profitable. Experimental analysis of these group scheduling strategies for the corresponding models delivered results consistent with the results of the profile data analysis.

The two scheduling strategies (chain and block) of this study were examined in isolation and treated separately. The block scheduling method occurred to us more as a generalization of the chain scheduling approach rather than a direct derivation from the profile data. However, the results with block scheduling have encouraged us to go back to the profile data for a deeper study of the available parallelism

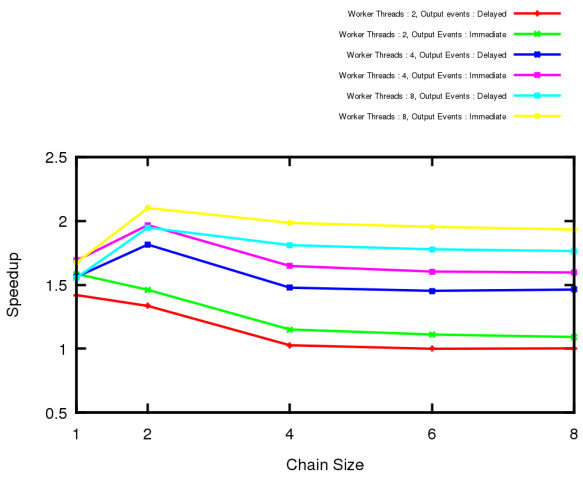

Figure 26: Overall Speedup of Epidemic using Event Chains

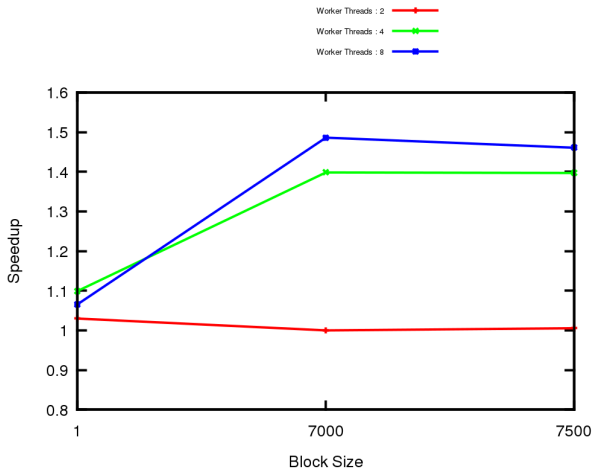

Figure 29: Overall Speedup of Epidemic using Block Scheduling

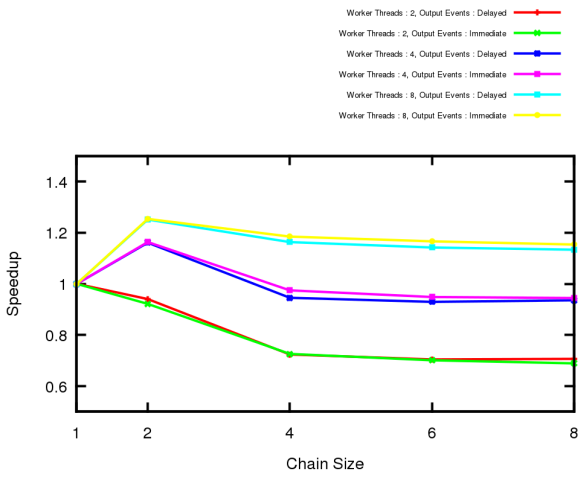

Figure 27: Relative Speedup of Epidemic using Event Chains

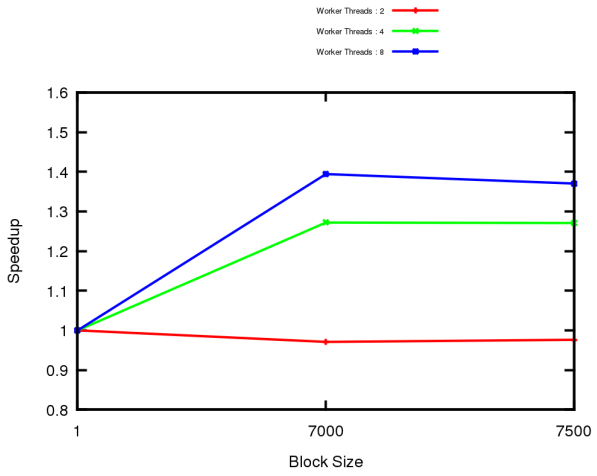

Figure 30: Relative Speedup of Epidemic using Block Scheduling

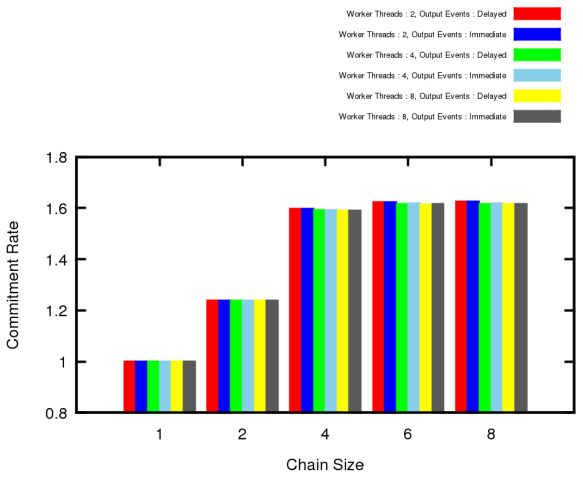

Figure 28: Commitment Rate of Epidemic using Event Chains

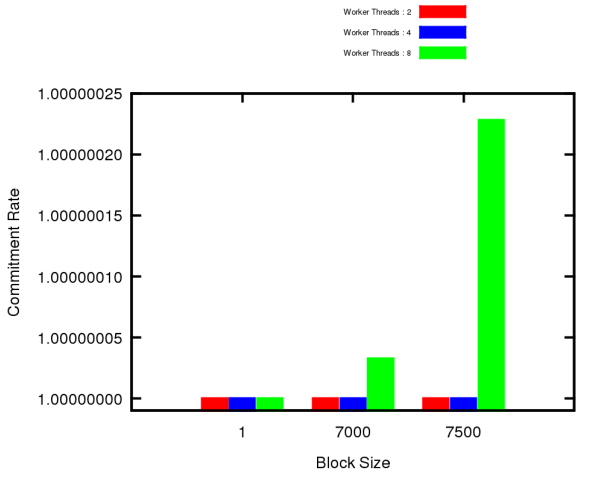

Figure 31: Commitment Rate of Epidemic using Block Scheduling

results. Our next question from this study is "Should we consider the average parallelism results *and* the chain results, to suggest that block and chain scheduling can be combined to achieve even better performance results?". This remains a question that we have yet to explore.

While the idea of scheduling multiple events during each scheduling event is not new and has already been explored by others, it is encouraging to have the performance results follow the profile data results. Ideally we will be able to discover additional new optimization strategies and techniques that are yet to be derived from a continued and extended profiling of simulation models.

8. ACKNOWLEDGMENTS

This material is based upon work supported by the AFOSR under award No FA9550–15–1–0384.

9. REFERENCES

[1] A. Alt and P. A. Wilsey. Profile driven partitioning of parallel simulation models. In *Proceedings of the 2014 Winter Simulation Conference*, pages 2750–2761, Savannah, GA, USA, 2014. IEEE Press.

[2] C. L. Barrett, K. R. Bisset, S. G. Eubank, X. Feng, and M. V. Marathe. Episimdemics: An efficient algorithm for simulating the spread of infectious disease over large realistic social networks. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC '08, pages 37:1–37:12, Piscataway, NJ, USA, 2008. IEEE Press.

[3] C. D. Carothers, D. Bauer, and S. Pearce. ROSS: A high-performance, low memory, modular time warp system. In *Proceedings of the Fourteenth Workshop on Parallel and Distributed Simulation*, PADS '00, pages 53–60, Washington, DC, USA, 2000. IEEE Computer Society.

[4] S. Das, R. Fujimoto, K. Panesar, D. Allison, and M. Hybinette. Gtw: a time warp system for shared memory multiprocessors. In *Proceedings of the 26th conference on Winter simulation*, pages 1332–1339, Orlando, Florida, USA, 1994. Society for Computer Simulation International.

[5] T. Dickman, S. Gupta, and P. A. Wilsey. Event pool structures for pdes on many-core beowulf clusters. In *Proceedings of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, pages 103–114, Montreal, Canada, 2013. ACM.

[6] R. M. Fujimoto. *Parallel and Distribution Simulation Systems*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1999.

[7] S. Gupta and P. A. Wilsey. Lock-free pending event set management in time warp. In *Proceedings of the 2nd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, pages 15–26, Denver, CO, USA, 2014. ACM.

[8] J. Hay and P. A. Wilsey. Experiments with hardware-based transactional memory in parallel simulation. In *Proceedings of the 3rd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, pages 75–86, London, UK, 2015. ACM.

[9] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5th edition, 2012.

[10] D. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):405–425, July 1985.

[11] Y.-B. Lin and P. A. Fishwick. Asynchronous parallel discrete event simulation. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 26(4):397–412, July 1996.

[12] D. E. Martin, T. J. McBrayer, and P. A. Wilsey. Warped: A time warp simulation kernel for analysis and application development. In *System Sciences, 1996., Proceedings of the Twenty-Ninth Hawaii International Conference on,*, volume 1, pages 383–386, Hawaii, USA, 1996. IEEE.

[13] K. S. Perumalla and S. K. Seal. Discrete event modeling and massively parallel execution of epidemic outbreak phenomena. *Simulation*, 88(7):768–783, July 2012.

[14] R. Radhakrishnan, D. E. Martin, M. Chetlur, D. M. Rao, and P. A. Wilsey. An object-oriented time warp simulation kernel. In *International Symposium on Computing in Object-Oriented Parallel Environments*, pages 13–23, Berlin, Germany, 1998. Springer Berlin Heidelberg.

[15] R. Rönngren, R. Ayani, R. M. Fujimoto, and S. R. Das. Efficient implementation of event sets in time warp. In *Proceedings of the Seventh Workshop on Parallel and Distributed Simulation*, pages 101–108, San Diego, California, USA, 1993. ACM.

[16] E. Santini, M. Ianni, A. Pellegrini, and F. Quaglia. Hardware-transactional-memory based speculative parallel discrete event simulation of very fine grain models. In *Proceedings of the 2015 IEEE 22Nd International Conference on High Performance Computing (HiPC)*, pages 145–154, Washington, DC, USA, 2015. IEEE Computer Society.

[17] S. J. Turner and M. Q. Xu. Performance evaluation of the bounded Time Warp algorithm. *Proceedings of the SCS Multiconference on Parallel and Distributed Simulation*, 24(3):117–126, 1992.

[18] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, June 1998.

[19] D. Weber. Time warp simulation on multi-core processors and clusters. Master's thesis, University of Cincinnati, Cincinnati, OH, 2016.

[20] P. A. Wilsey. Some properties of events executed in discrete-event simulation models. In *Proceedings of the 2016 annual ACM Conference on SIGSIM Principles of Advanced Discrete Simulation*, pages 165–176, Banf, Alberta, Canada, 2016. ACM.