Some Properties of Events Executed in Discrete-Event Simulation Models

Philip A. Wilsey
Dept of EECS, PO Box 210030
University of Cincinnati
Cincinnati, OH 45221–0030
wilseypa@gmail.com

ABSTRACT

The field of computer architecture uses quantitative methods to drive the computer system design process. By quantitatively profiling the run time characteristics of computer programs, the principal processing needs of commonly used programs became well understood and computer architects can focus their design solutions toward those needs. The DESMetrics project is established to follow this quantitative model by profiling the execution of Discrete Event Simulation (DES) models in order to focus optimization efforts within DES execution frameworks (and especially parallel DES engines). In particular, the DESMetrics project is designed to capture the run time characteristics of event execution in DES models. Because DES models tend to have fine grained computational processing requirements, the DESMetrics project focuses on the event dependencies and their exchange between the objects in the simulation. For now, we assume that optimization of the actual event processing is well served by conventional compiler and architecture solutions. Although, as will become clear later in Section 6, the possibility of identifying scheduling blocks of events that could potentially be schedule together can be achieved — at least within a single simulation object.

CCS Concepts

•Computing methodologies → Modeling and simulation; Discrete-event simulation; *Simulation tools;*

Keywords

Discrete event simulation; Profiling simulation models; Parallel discrete event simulation

1. INTRODUCTION

The field of computer architecture has been dramatically impacted by the use of quantitative methods to drive the computer system design process [12]. By quantitatively profiling the run time characteristics of computer programs, the principal processing needs of commonly used programs became well understood and computer architects can focus their design solutions toward those needs. This has resulted in the widespread availability of high performance computing in the commodity processor and optimizing compiler markets.

This paper introduces a project to collect quantitative data from Discrete Event Simulation (DES) models in order to better understand their (primarily) computational needs. The primary motivation for this work is to better understand the properties of discrete event simulation models in order to pursue a more focused effort to build high-performance Time Warp [13, 11] synchronized parallel simulation engines. The project, called *DESMetrics*, follows the quantitative methods used by the architecture community. In particular, we instrument discrete event simulation engines to record (to a file) information on the events processed. This file is then analyzed to produce `csv` files that can be analyzed and displayed. Because DES models tend to have fine grained computational processing requirements, the DESMetrics project focuses on the event dependencies and their exchange between the objects in the simulation. For now, we assume that optimization of the actual event processing is well served by conventional compiler and architecture solutions.

In this paper, we present the processes and tools used in the DESMetrics project to collect and display this information. Furthermore, the event properties from several discrete event simulation models are captured and reported. In particular, two different discrete event simulation engines are instrumented and data collected. Two simulators are used in order to capture data from simulation models written by different authors and processed by different simulators. The purpose is not to compare the different simulators or to establish performance comparisons between the simulators. Instead the goal is to capture properties of various different simulation models in order to extract common properties from the simulation models that could serve to direct a focused optimization effort for some simulation engine. The chief contribution in this manuscript is to outline a general framework for data capture, analysis, and visualization and to demonstrate that different simulation models do in fact have common characteristics that can aid in optimization studies.

The DESMetrics project focuses strictly on the relations between events and the processes that process these events. By limiting the focus in this way, we have found it fairly easy to instrument a discrete event simulation engine to capture this data. Using this data, a variety of different analysis steps can be pursued. Of particular focus in this paper is: (i) the potential parallelism available, (ii) the number of LPs that each LP receives events from, (iii) the uniformity of events processed by the different objects of the simulation,

SIGSIM-PADS '16, May 15-18, 2016, Banff, AB, Canada
© 2016 ACM. ISBN 978-1-4503-3742-7/16/05. . . $15.00
DOI: http://dx.doi.org/10.1145/2901378.2901400

and, (iv) the length of events in input queues that can be scheduled for execution from a fixed time point. This latter measure is called an *event chain* and will be described more formally in Section 6.3.

The remainder of this paper is organized as follows. Section 2 presents some background information. Section 3 reviews some previous work related to this paper. Section 4 gives a high level overview of the DESMetrics process and tools. Section 5 presents the simulation kernels and simulation models studied and reported herein. Section 6 presents the quantitative data captured from these simulation models. Readers are cautioned that several of the graphs in this section have significant data points and may take a minute or two to render, even on a higher performing desktop platform. While it is possible to reduce the resolution of these graphs for faster rendering, preserving the detail permits the interested reader to zoom in and observe the detailed results. As a result, we have preserved the detail. Finally, Section 7 contains some concluding remarks.

2. BACKGROUND & MOTIVATION

Capturing and understanding the properties of DES models can aid researchers in a variety of ways. The obvious aid is in providing guidance to areas to focus on performance improvement in simulation kernels (parallel and otherwise). To a certain extent, the focus of kernel optimization can be (and has been) by profiling the simulation kernel with tools like `valgrind`. However, that will only inform the developer of bottlenecks in an existing code base. Using results from the quantitative properties of the application domain directs the optimization focus on algorithm development for the key model specific properties and not (solely) toward improving the implementation of the algorithm.

A second area where a deeper understanding of the properties of DES models is in synthetic workload generation. A substantial amount of work has been directed at generating synthetic simulation models to exercise parallel simulation kernels [2, 7, 9, 18]. These synthetic models are commonly used by the research community to evaluate and report performance results. However, the utility of the specific configurations used to derive these synthetic workload generators is often not well established. Using profile data from actual models could dramatically improve confidence in the models generated for synthetic testing.

Related to algorithm optimization, simulation model profile data can sometimes be used to enable model setup and configuration for more optimal simulation. In fact, the trace data used from the WARPED2 simulation kernel [21] and used in this study is actually captured by translating information already captured by the kernel to perform LP partitioning [1].

Finally, the use of profile data could potentially be used to aid in the validation and verification of a simulation model (at a very coarse level). For example, simulation models with a scale-free network should have a specific communication profile between the objects of the simulation. Visualizations of the event connectivity between LPs could help confirm that the network does indeed follow the shape of communication expected.

3. RELATED WORK

Some of the earliest work analyzing the properties of discrete event simulation address, for example, the amount of parallelism available or the lookahead possible in simulation models. Early studies to perform *critical path analysis* [4, 14, 17, 15] are an attempt to locate the shortest path through the collection of events in a discrete event simulation model. Two of these studies develop various algorithms that can be embedded in sequential simulators to locate the critical path [17, 15]. These studies can be used to show the fastest path to completion through the event pool of a simulation. Another common property of discrete event simulation analysis is to evaluate the lookahead available in simulation models [10]. Lookahead plays an important role in determining if a simulation model is suitable for parallelization using conservative synchronization techniques. This work extends those efforts by studying multiple properties of discrete event simulation models and working to understand how these properties could potentially be exploited to improve the performance of discrete event simulation engines — parallel and sequential.

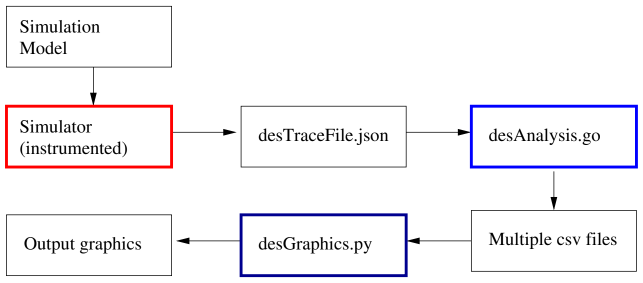


Figure 1: The DESMetrics Tool Flow

4. THE DESMETRICS PROCESS

The approach in the DESMetrics project is to capture event information from existing simulation models by instrumenting DES simulators to capture said information. The captured information is then analyzed to produce summary data files that can then be visually displayed for inspection. This process is visualized in the graphic of Figure 1. Currently we have instrumented two different simulators (ROSS [5] and WARPED2 [21]) to capture the event information (only the sequential versions of both kernels can be used for this capture). These simulators are depicted by the red box in Figure 1. Analysis on the captured event data is performed by a `go` program named `desAnalysis.go` and depicted by the light blue box in Figure 1. This analysis produces a number of `csv` files containing the analysis results. Finally, a set of graphs are produced by a python/matplotlib program named `desGraphics.py` and depicted by the dark blue box in Figure 1. The next three subsections describe each of the main processing steps in the DESMetrics project.

4.1 Recording the Event Data

Event data is captured by instrumenting existing simulators to produce an event trace file. The capture process assumes that each simulator maintains a simulation time value and that the simulation is organized such that events are processed and exchanged by named simulation objects. The naming of the simulation objects is not critical and the names can be anonymously generated, however to facilitate analysis, the tools assume that events are exchanged by simulation objects and that the names are unique to the simulation objects in the simulation model. In the remainder of this paper we will use the term Logical Process or LP (from parallel simulation [8]) to denote a simulation object that processes and exchanges events.

```
{
"simulator_name" : "name of the simulator",

"model_name" : "name of the simulation model",

"capture_date" : "date/time that the profile data
     was captured",

// optional, but desirable; include if possible
"command_line_arguments" : "significant command
    line arguments",

// optional, include as needed
"configuration_information" : "anything
    significant",

"events" : [
  ["source object",  send_time, "destination
      object", receive_time ],
  //  "....forall events processed...."
  ]
}
```

Figure 2: Format of JSON file holding simulation model trace data

The event information is captured in a `json` file format. Since even small simulation runs can easily produce gigabytes of event data, the `json` file format is somewhat more compact than might be expected. However, experience with more verbose formats resulted in file sizes that were quite difficult to process and therefore this compacted form is now used. The general format for captured data is (with non-json compliant comments added) shown in Figure 2. The fields of this format are defined as: the `send_time` is the simulation time when the event was generated and `receive_time` is the simulation time when the event is to be executed; the `source object` and `destination object` are the names of the LPs that, respectively, generate and process the event. Note that events sent by an object to itself are, if available, recorded as well as events exchanged between objects. Additional information on the event payload is not necessary and not captured.

4.2 The Analysis Phase

The analysis phase is performed by the `desAnalysis.go` program. This program takes considerable time (for example analyzing a 5GB file can take 10-15 hours on an 4-core/8-thread i7 x86 processor). It has been parallelized to optimize run time performance on multi-core processors. The analysis phase creates a collection of `csv` files that are then read by the tools in the visualization phase to produce plots (`pdf` or `eps`) for viewing.

The analyses performed in this phase are organized into the following classes:

Events Available for Execution: assuming unit time execution of each event, how many events are available at any given time for execution? This analysis attempts to compute a *conservative* estimate of how much potential parallelism exists among the events and as such, optimistic execution could easily uncover. The specific algorithm to compute events available is given in Section 6.1.

Events Executed by LP: how many events does each LP execute? Are the events self-generated (called *local* events) or remotely generated from some other LP (called *remote* events)?

Event Chains: are chains/blocks of events that could potentially be executed as a group from a fixed simulation time. That is, at a specific simulation time, how many events stored at that time would be available for immediate execution without the LP receiving any additional information? The algorithm computes the number of chains of various lengths. The details of this computation are given in Section 6.3.

Event Exchanges between LPs: for each LP, from how many different LPs does it receive events for execution? This analysis attempts to illustrate the degree of connectivity of events exchanged by the LPs in the simulation.

Lookahead data: for each LP, the analysis captures the delta of the send and receive timestamp of events exchanged between two LPs. The minimum, maximum, and average of this delta is captured. This data is captured separate for local and remote events as the lookahead information is critical only for event information exchanged between LPs on different compute nodes.

A detailed description of the analysis for above classes is described in the subsections of the results section (Section 6) below.

4.3 Visualizing the Analysis Results

The visualization phase is performed by the `desGraphics.py` program. This is a python program that uses `matplotlib` and `pylab` to produce `pdf` or `eps` files for visualization of the results.

5. SIMULATION MODELS STUDIED

The two simulators studied are ROSS [5] and WARPED2 [21]. A fork of the instrumented ROSS code base is available in the git repository https://github.com/wilseypa/ROSS. The WARPED2 code base (available at https://github.com/wilseypa/warped2) already captures the necessary event traces so only a short translation script is needed to convert the data into the desired format (available in the desMetrics code base). All of the tools for the DESMetrics project are release with open source licensing and available from the git repository https://github.com/wilseypa/desMetrics. The data files are available but their size (several GB each) prevent their online distribution.

In this paper, we report results from 4 simulation models, two from ROSS and two from WARPED2. All of the simulation models studied were taken from the standard code base of these tools. No modifications were made to the simulation models. The models used are:

traffic: a 2-d model of automobile traffic simulation model (ROSS).

pcs: wireless network model(s). The PCS model from ROSS is described in [6] and the PCS model from WARPED2 is based on the model described in [16]. The ROSS model uses a exponential distribution for event distribution; the WARPED2 model uses a Poisson distribution.

epidemic: an disease propagation model in WARPED2 derived from [3, 19].

In general the default configuration parameters for these models were used. However, in all cases, a shorter simulation time than the default (if one existed) was required. The command line used to capture these instrumented data for each simulation models is:

ROSS:

```
./pcs --synch=1 --end=10000
./Intersection --synch=1 --end=25
```

WARPED2:

```
./pcs_sim --statistics-type csv
  --statistics-file desMetrics.csv
  --max-sim-time 500
./epidemic_sim --statistics-type csv
  --statistics-file desMetrics.csv
  --max-sim-time 1000000
```

Algorithm 1: Compute the number of events available at each simulation cycle.

```
Input  : LP[] array of all LPs, where LP[i].event_queue
         denotes the queue of events destined for LP[i]
Output: events_available[], counts of events available at each
         simulation cycle i
begin
    total_schedule_cycles ← 0
    forall the i in 1:N do
        events_available[i] ← 0
    end

    while (at least one LP[i].event_queue.empty() != NULL)
    do
        Set schedule_time to the lowest receive_time in the LP
        array.
        schedule_time ←
        minimum(LP[i].event_queue.front().receive_time)
        Count the number of LPs with events that were sent
        before schedule_time
        for each i such that
        (LP[i].event_queue.front().send_time < schedule_time)
        do
            events_available[schedule_time]++
            LP[i].event_queue.pop()
        end

        total_schedule_cycles++
    end

    for i in range (1:schedule_time) do
        plot i,events_available[i]
    end

end
```

6. RESULTS

The results are presented in five separate subsections. The first discusses results from the "*events available for execution*" portion of the analysis phase. This analysis orders the events by receiving LP and performs a simulated walk through an execution of the events. The second subsection discusses the nature of *events executed by the LPs* in the running simulation. The primary objectives here are to show how many events are executed by the LPs and to classify events that are generated locally (by the executing LP) or remotely (by some other LP). Note that not all simulators will have locally generated events (*e.g.*, MiniSSF [20]), but in this study, both kernels do have local events. The third subsection studies *event chains*. Event chains are blocks of events in the pending event set that can potentially be scheduled as a block chain of events. Such chains are important to a parallel simulation engine on an SMP platform. More specifically if block chains of lengths greater than 1 are commonly present in simulation models, then a block scheduling of events can help reduce contention for the shared data structures maintaining the pending event set. The fourth subsection contains a review of our findings on the number of LPs that each LP receives remote events from. Finally, the fifth subsection contains a brief review of results regarding lookahead results.

6.1 Events Available for Execution

The computation of events available is developed to better understand the maximum potential parallelism in the simulation that guarantees safe execution of all events. The computation basically runs a simulated event execution engine to evaluate the number of events available at every simulation cycle (for simplicity, we assume that events available for execution are simulated instantaneously). While this sets an upper bound for parallelism in conservatively synchronized parallel simulation, it does not necessary find all of the parallelism that might be uncovered in an optimistically synchronized parallel simulation. The remainder of this section is subdivided into two parts. The first part presents the algorithm used to compute events available. The second part presents the principal findings with the simulation models studied.

Computing Events Available for Execution

The computation of events available performs a conservative estimate of what LPs have events that can be guaranteed to be safe for parallel execution. Essentially the computation orders the LP event queues by their `receive_time` and moves to the first event at each LP, called the *head event*. The head event with the lowest timestamp defines the *evaluation time*. The algorithm then counts as available every LP whose head event has a `send_time` before the evaluation time and a `receive_time` at or after the evaluation time. The head event for all counted LPs is advanced to the next event and the process is repeated until no more events exist in the set of LPs. For each LP report the number of local events executed and the total (local plus remote) number of events executed. The pseudo code of the algorithm used do perform this analysis is given in Algorithm 1.

The events available data computed by Algorithm 1 has several more steps and records several other data points of interest. At completion of this analysis phase, the results are dumped into several `csv` files that can then be processed for visualization or other analysis steps.

Results from Events Available Study

The plots in Figure 3 show the number of events available by simulation cycle. The label on the top of the graph shows the total number of LPs in the simulation, the bottom x-axis shows the number of simulated simulation cycles and their total. The left y-axis shows the raw number of events available at each simulation cycle

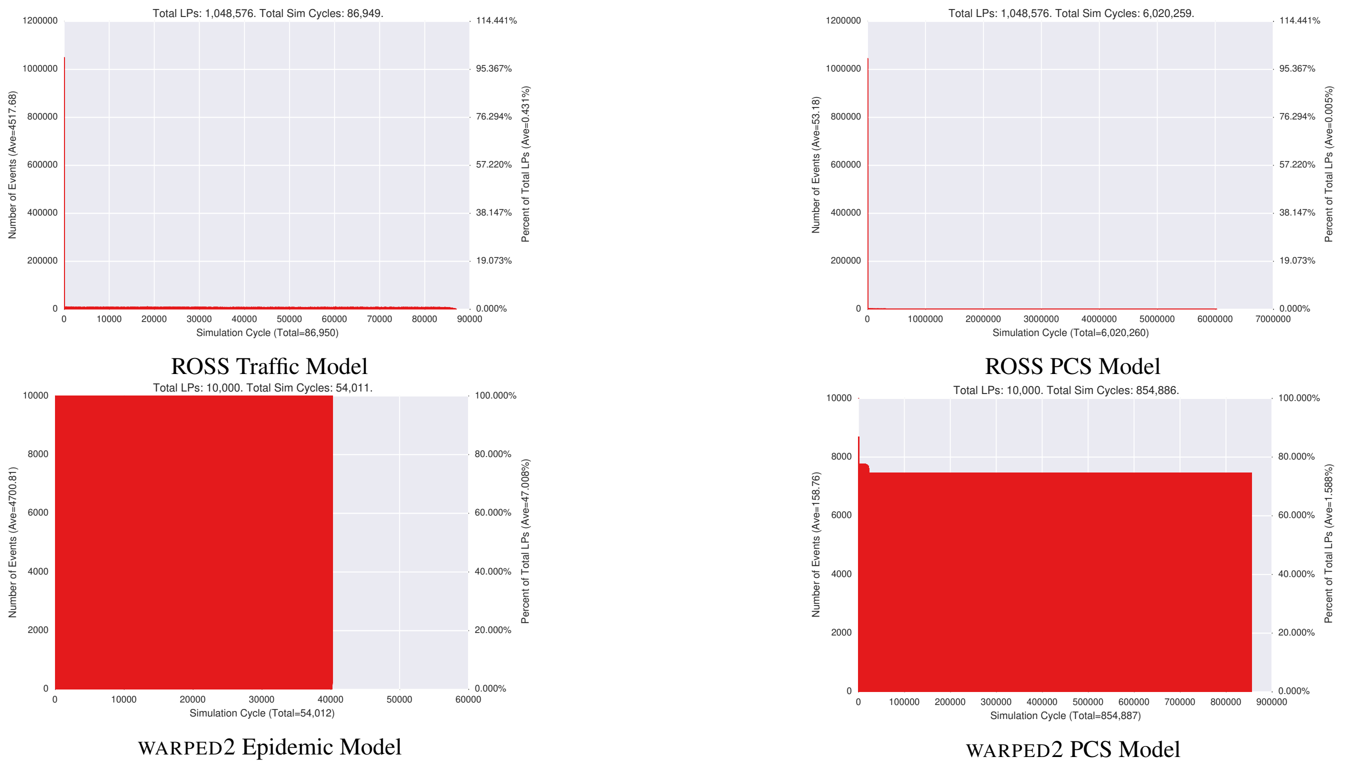


Figure 3: Events available for execution at each simulation cycle.

and the average over all of the cycles. The right axis is a scale of the events available as a percentage of the total LPs (and the average). That is, what percentage of LPs had an event available at that simulation cycle.

Unfortunately for the two ROSS models, there are an unusually large number of events available at simulation startup and teardown (not visible in the graphs); To a lesser extent, this also occurs in the WARPED2 PCS model. Since we are really trying to discover the "common case" of processing requirements, a more instructive visual might be to examine the data with outliers removed. Initially we developed plotting scripts that removed all data points that were greater than 2 standard deviations from the mean (Figure 4). This worked reasonably well, however, we found that trimming the first and last 1% of the evaluation cycles also achieved the desired effect. The advantage of this approach is that it attacks startup and teardown costs without discarding any wild variations that might exist during the main portions of the simulation. These results are shown in Figure 5.

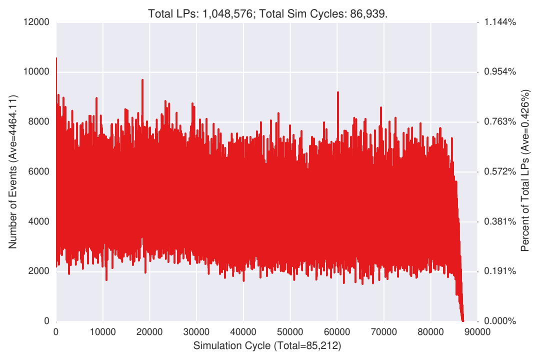
Examining the results in Figure 5, we note that there is wide swings in the number of events available (*e.g.,* ROSS Traffic swings roughly between 2,000 and 8,000 events per simulation cycle). In terms of raw averages, we note that a only moderate percentage of LPs in the simulation have events available for concurrent execution (0.005%–47.0%). However, for any reasonably large simulation there are more than sufficient events available (52–4,700) for concurrent execution for a moderately sized parallel processing platform.

Figure 6 contains a histogram of the number of events available in the simulation. The x-axis is the number of events available, the left y-axis shows the number of simulation cycles, and the right axis shows the scale of percentage of the total simulation cycles. It is interesting to note that both PCS models have a large number of simulation cycles with only a very few events available (indicating a relatively low degree of parallelism). In contrast, both Traffic and Epidemic have histogram maximums where the number of events available are in the thousands.

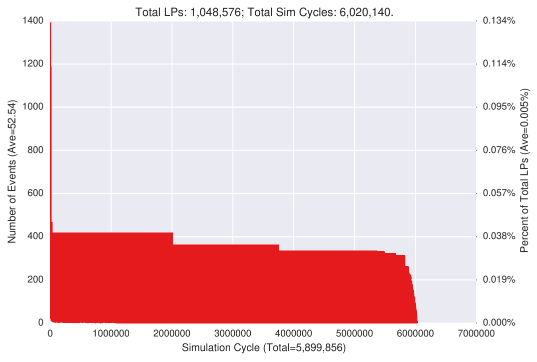
6.2 Profile of Events Executed by the LPs

During the events available analysis step, the DESMetrics tools also separates and records the events executed by each LP into two classes that we call *Local* and *Remote*. Local events are events that were generated and processed by the same LP. Remote events are generated by one LP and processed by another LP. As previously noted, not all simulation engines will send local events. However, both ROSS and WARPED2 do generate local events and the result of this analysis may be helpful for designing and optimizing pending event scheduling algorithms.

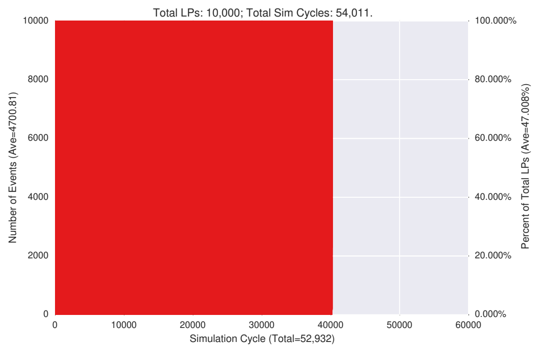
Figure 7 contains the results of events executed by LP. In this case, the LPs are sorted by the total number of events processed by it in the entire simulation (the blue line). The red line in these graphs shows the raw number of events that are locally generated. The green line shows the percentage of events that are locally generated. Except for ROSS Traffic, the percentage of events that are locally generated is quite high (averaging well over 50%). In the context of optimizing a parallel simulator, one could potentially create algorithms that short circuits the placement of a newly generated event into the input queue and, in some cases, simply directly executes it. This would bypass the locking overheads of accessing shared data structures of the pending event set and potentially have significant positive performance implications. This concept will be re-examined in the event chains discussion of the next section.



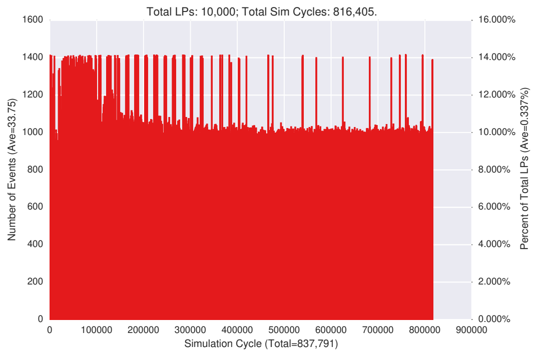
ROSS Traffic Model



ROSS PCS Model

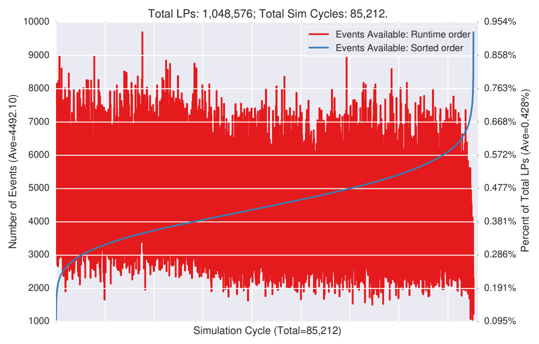


WARPED2 Epidemic Model

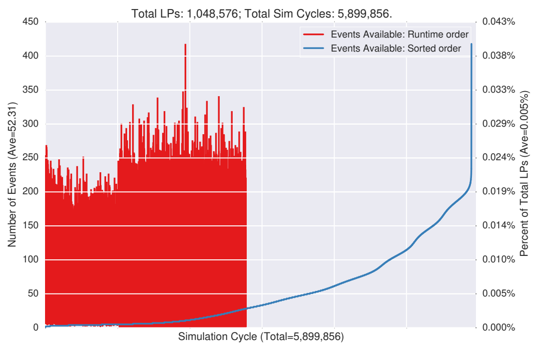


WARPED2 PCS Model

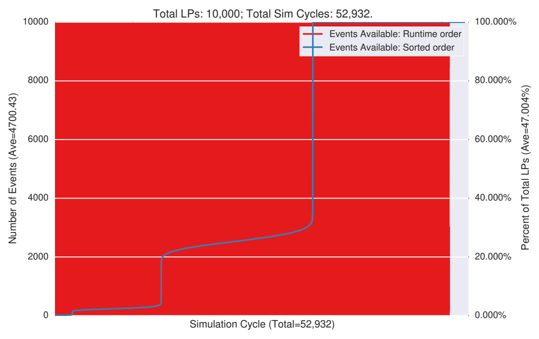
Figure 4: Events available for execution at each simulation cycle (outliers of $\sigma \geq 2$ removed).



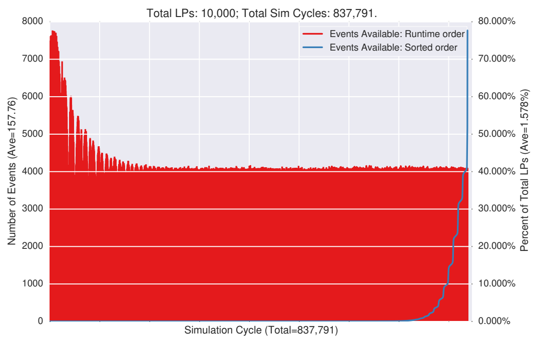
ROSS Traffic Model



ROSS PCS Model

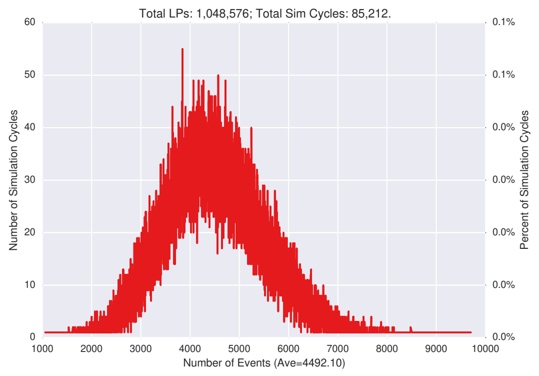


WARPED2 Epidemic Model

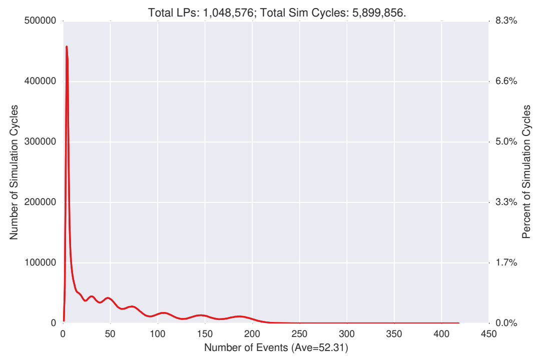


WARPED2 PCS Model

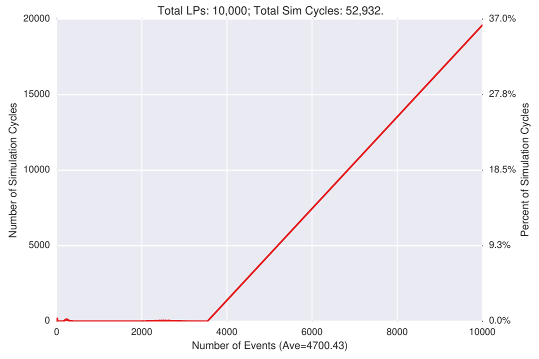
Figure 5: Events available for execution at each simulation cycle (First and Last 1% of simulation cycles removed).



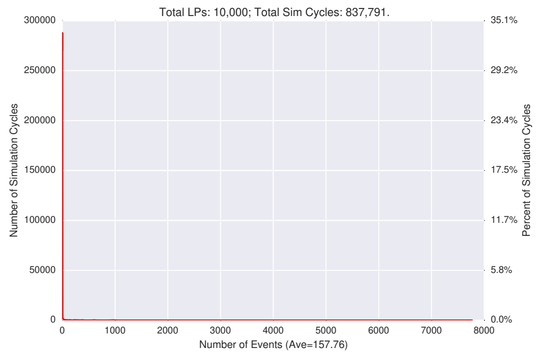
ROSS Traffic Model



ROSS PCS Model

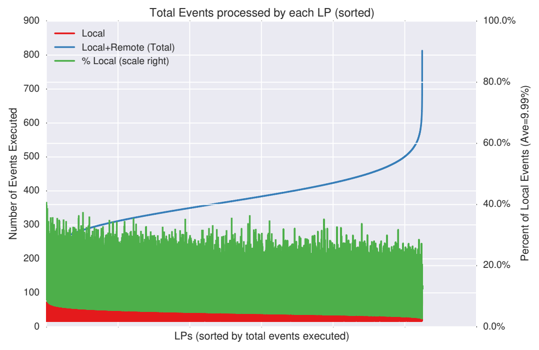


WARPED2 Epidemic Model

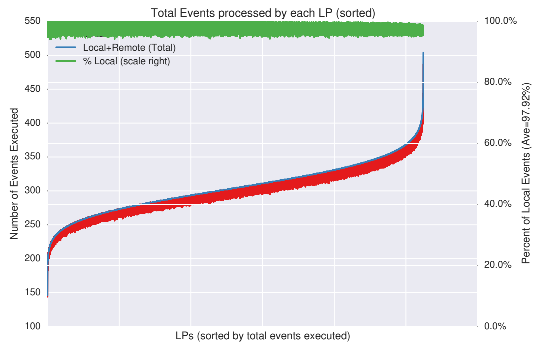


WARPED2 PCS Model

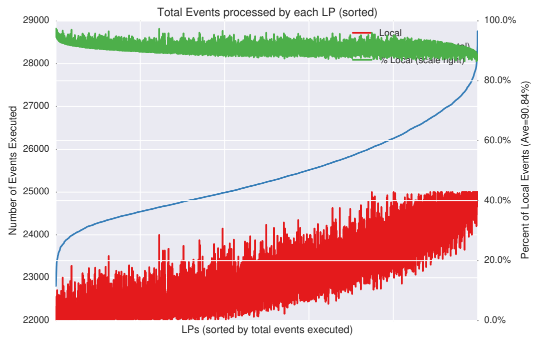
Figure 6: Histogram of events available for execution at each simulation cycle (First and Last 1% of simulation cycles removed).



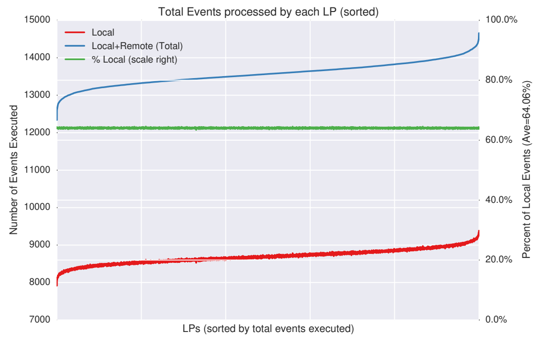
ROSS Traffic Model



ROSS PCS Model

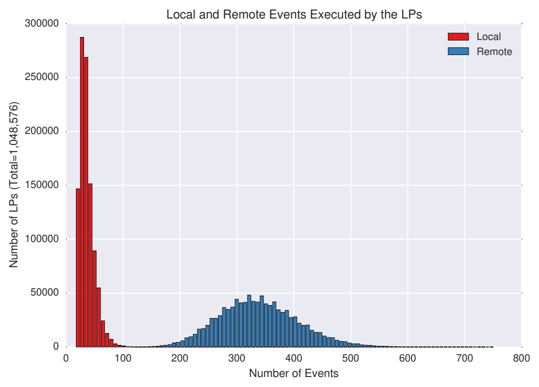


WARPED2 Epidemic Model

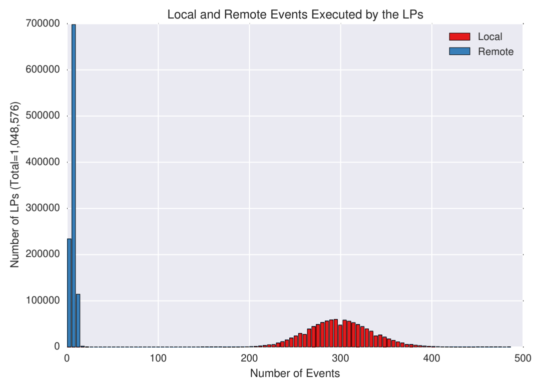


WARPED2 PCS Model

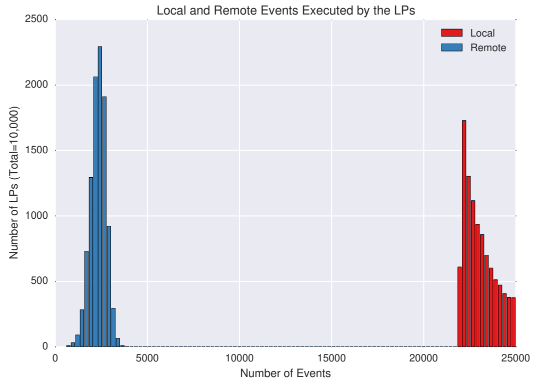
Figure 7: Total events executed by each LP (for all simulation cycles.



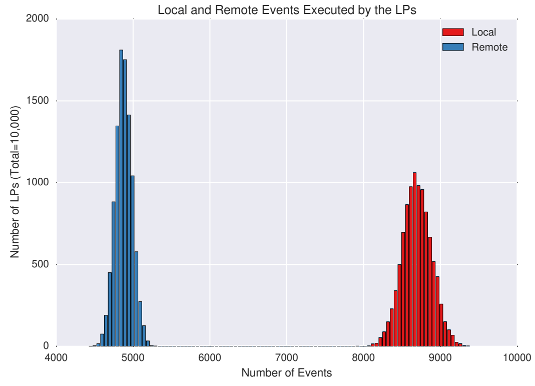
ROSS Traffic Model



ROSS PCS Model



WARPED2 Epidemic Model



WARPED2 PCS Model

Figure 8: Histogram of events executed by each LP by event class (Local or Remote).

Figure 8 presents the data on local/remote events executed by LPs using the stacked histogram plotting capabilities of matplotlib. This graph summarizes the number of local/remote events executed by the LPs. In this graph, the x-axis shows the number of events executed (red — local, blue — remote) and the y-axis shows the number of LPs that have said number of events.

6.3 Event Chains

Event chains are blocks of events in the pending event set of an LP that could potentially be executed together. This computation is established independently of the events available execution loop shown in Algorithm 1. Instead, we examine the events processed by each LP independently. At each step, a chain is constructed and its maximum length counted. All of the events in the 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. Formally, we define event chains from the `receive_time` timestamp of the head event in the chain. Thus, let e_0 denote the head event and e_i denote the event being considered for inclusion in the chain. Chain membership is established if

$$e_0.receive\_time > e_i.send\_time.$$

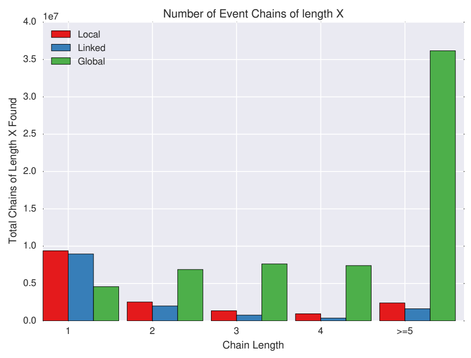
Event chains are further classified into three types, namely: *local*, *linked*, and *global*. To be members of local or linked chains, events must be locally generated events (generated by the executing LP). Membership in global chains places no constraint on which LP generates the event.

A *linked* chain is similar to the local chain except that the constraint on the `send_time` is relaxed so that any event with a `send_time` less than an event executed in the chain is also determined to be a member of the chain. Thus, any event generated by the chain is also potentially a member of the chain. That is, the linked chain begins with the list of events in the local chain and then includes any event with a send time that is less than the receive time of the last event in the chain. Events added to the chain can thus trigger the addition of yet more events into the chain.

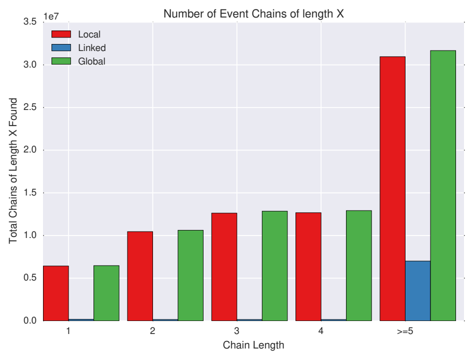
In the DESMetrics implementation of event chain analysis, only chains up to a maximum length of 4. All chains longer than this limit are counted together and labeled ≥ 5.

Figure 9 shows the number of chains on the y-axis given their length shown on the x-axis. Note that the y-axis labels are in millions. Results for local (red), linked (blue), and global (green) are shown. Note that these counts are of the chains found corresponding to that length and the counts are not inclusive. To expand on this, Figure 10 shows the counts as a cumulative total of all chains at or greater than this length. What is interesting in these graphs is that sizable number of chains of length greater than one present in some of these simulation model. This suggests that block scheduling of events may be an effective optimization strategy for an Time Warp synchronized parallel simulation kernel.

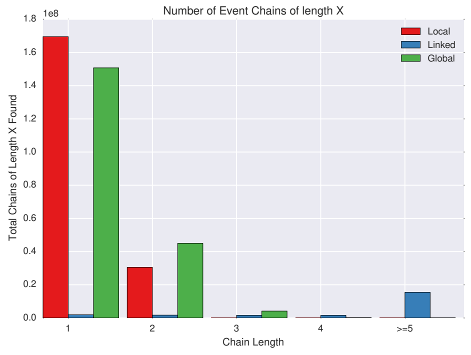
Block scheduling of events by a simulator means that the event processing loop would dequeue and process multiple events from a single LP as a group. This could be especially beneficial for a kernel executing multiple event processing thread on an SMP machine. The key benefit to block scheduling is reduced time in the critical



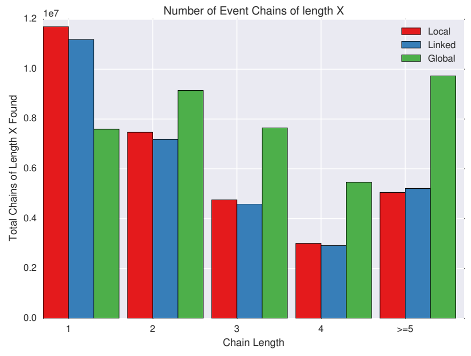
ROSS Traffic Model



ROSS PCS Model



WARPED2 Epidemic Model



WARPED2 PCS Model

Figure 9: Number of event chains of length n.

region of the pending event set. Of course block scheduling might also lead to premature computation and thereby necessitate rollback. However, block scheduling of local event chains could potentially reduce the premature computation prospect. In any case, the data suggests it may be worth examining as a potential strategy for optimizing an SMP based platform simulation kernel.

6.4 Event Exchanges between LPs

In this analysis, we are attempting to discover how many LPs each LP sends events to. In order to better understand the connectivity of event exchanges among the LPs, this analysis counts the number of LPs that send an event to each LP. Basically this counts the number of LPs that send remote events to an LP. In addition to counting the total number of LPs that send a remote event, the analysis also computes how many events sends 95%, 90%, 80%, and 75% of the remote events. This allows us to better understand high connectivity and low connectivity of event exchanges among the LPs.

The results of this analysis are shown in Figure 11. In these graphs, the number of LPs that an LP receives remote events from is plotted on the y-axis. The data for each percentage shown is sorted (independently of the others) and then plotted on the same graph. For all of the simulation models, the number of sending LPs is a very small fraction (less than 1%) of the total LPs. While this does not show who is sending and therefore allows for the possibility that a small set of LPs communicate with all other LPs, our studies with partitioning of simulation models for parallel execution show that communication is distributed throughout the simulation model and that partitioning can largely localize communication between LPs [1]. As yet we have not discovered a better way to show this result from the analysis performed here, we strongly believe that effective partitioning can significantly improve performance of a parallel simulator. In fact, the WARPED2 simulator has profile driven partitioning which shows significant performance implications for all of these simulation models.

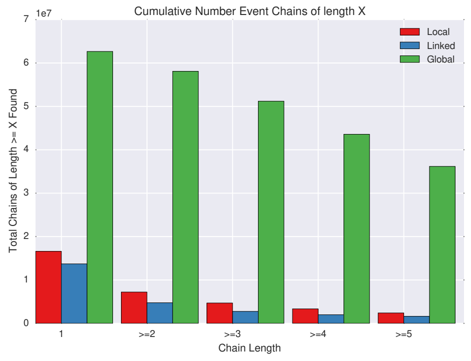
6.5 Lookahead

In the final step of the analysis reported in this paper, we examine information related to lookahead. Specifically we plot results showing the minimum and average timestamp delta ($receive\_time - send\_time$) of remote events sent by each LP (Figure 12). The minimum timestamp is effectively the guaranteed safe lookahead on the channel of events sent between two LPs. The data shows that for the models studied, only the WARPED2 epidemic model has any significant lookahead.

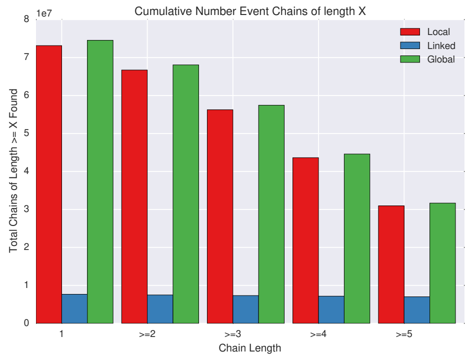
7. CONCLUSIONS

This paper presented an approach for capturing simulation time properties of events exchanged and executed in a discrete event simulation model. The approach is to instrument a discrete event simulation engine to capture profile data. The profile data is then analyzed to produce various relations between the events and the LPs processing events in the simulation. The principal goal for this work is to help direct the algorithm development for investigations into solutions with parallel simulation. We believe that these results can directly impact the research directions of research in parallel simulation.

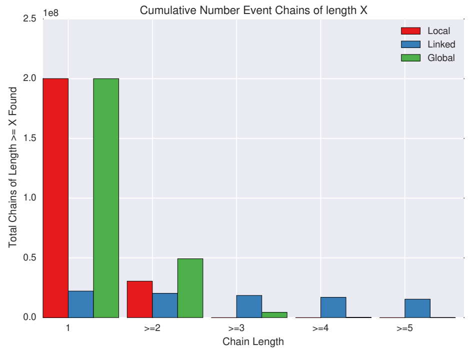
In summary, the data collected shows that ample parallelism exists in discrete event simulation models for parallelism to be successful. Furthermore, we find that simulation events are often lo-



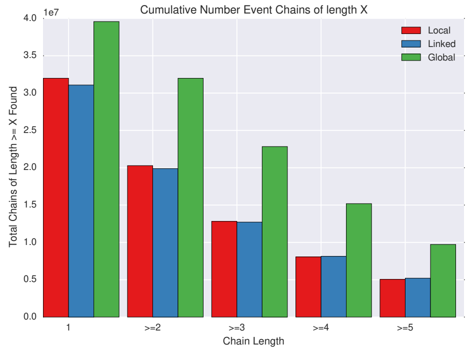
ROSS Traffic Model



ROSS PCS Model

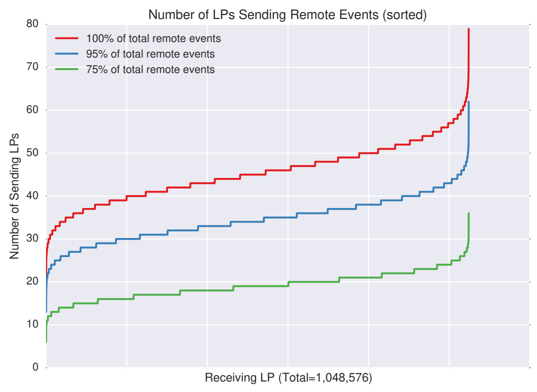


WARPED2 Epidemic Model

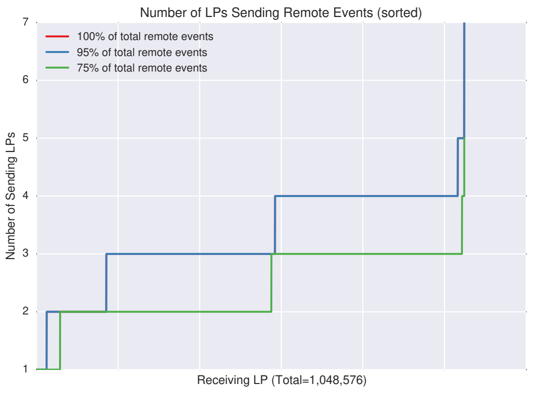


WARPED2 PCS Model

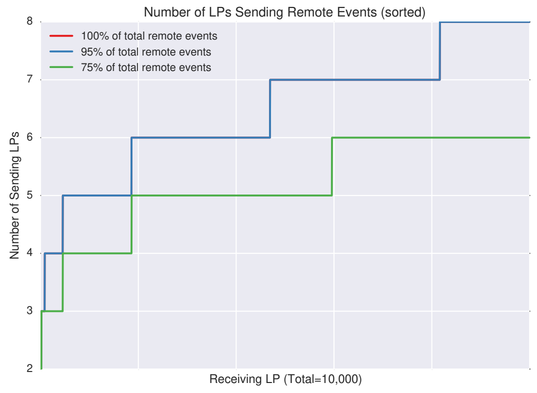
Figure 10: Cumulative number of event chains of with length $\geq$ n.



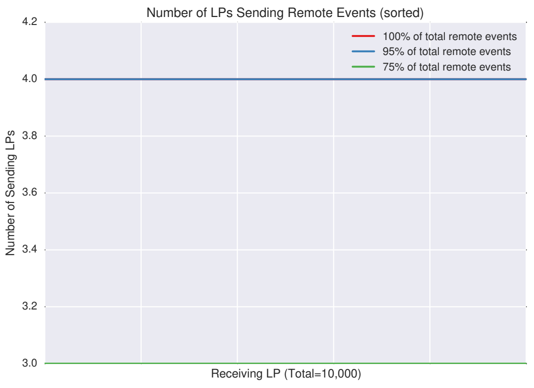
ROSS Traffic Model



ROSS PCS Model

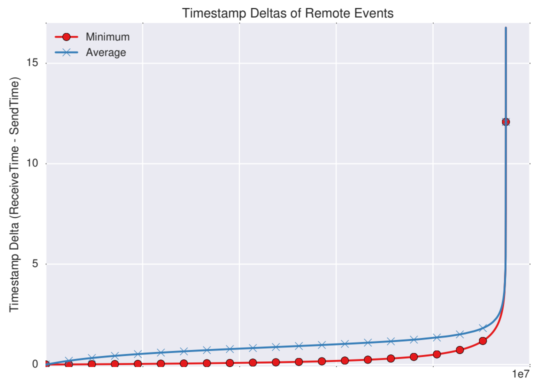


WARPED2 Epidemic Model

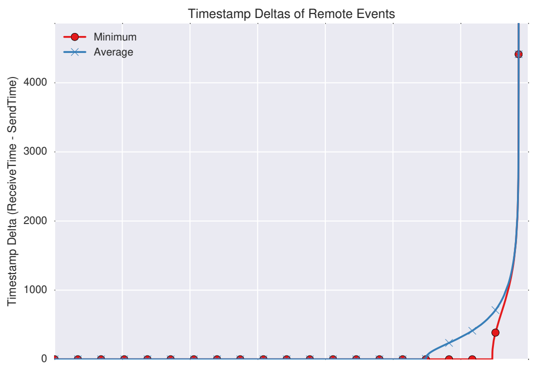


WARPED2 PCS Model

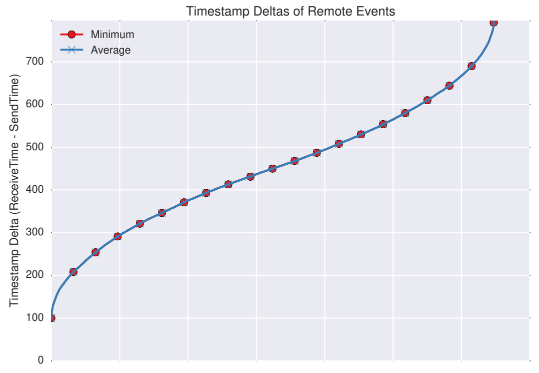
Figure 11: Number of LPs sending various percentages of remote events



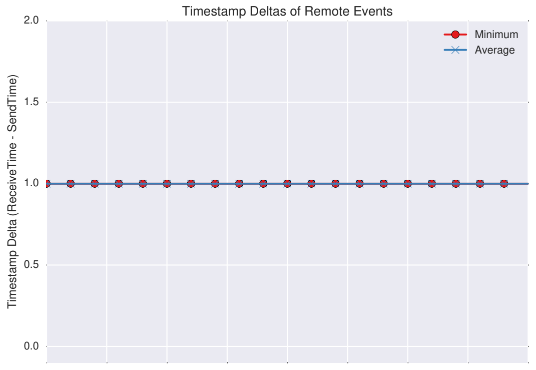
ROSS Traffic Model



ROSS PCS Model



WARPED2 Epidemic Model



WARPED2 PCS Model

Figure 12: The Delta of the Send and Receive time of Remote Events (Lookahead).

cal within an LP and that "gang" scheduling of events in an LP should be a highly effective technique to dramatically improve performance on a shared memory platform. Finally, the impact of effective partitioning of the simulation model cannot be sufficiently emphasized. To fully unlock the potential of parallel simulation, the PDES field must embrace static analysis techniques to help organize the simulation model for high performance. With the proper application of static analysis techniques, we believe that parallel simulation can have significant performance impact on the execution efficiency of large simulation models.

8. ACKNOWLEDGMENTS

Support for this work was provided in part by the National Science Foundation under award CNS–0915337 and this material is based upon work supported by the AFOSR under award No FA9550–15–1–0384.

9. REFERENCES

[1] A. J. Alt and P. A. Wilsey. Profile driven partitioning of parallel simulation models. In *Proceedings of the 2014 Winter Simulation Conference*, Dec. 2014.

[2] V. Balakrishnan, R. Radhakrishnan, D. M. Rao, N. B. Abu-Ghazaleh, and P. A. Wilsey. A Performance and Scalability Analysis Framework for Parallel Discrete Event Simulators. *Simulation Practice and Theory*, 8:529–553, 2001.

[3] 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, Piscataway, NJ, USA, 2008. IEEE Press.

[4] O. Berry and D. Jefferson. Critical path analysis of distributed simulation. In *Distributed Simulation*, pages 57–60. Society for Computer Simulation, 1985.

[5] 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.

[6] C. D. Carothers, R. M. Fujimoto, Y.-B. Lin, and P. England. Distributed simulation of large-scale PCS networks. In *Proceedings of the Second International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '94)*, pages 2–6, Jan. 1994.

[7] A. Ferscha and J. Johnson. A testbed for parallel simulation performance predictions. In *1996 Winter Simulation Conference Proceedings*, December 1996.

[8] R. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, Oct. 1990.

[9] R. Fujimoto. Performance of Time Warp under synthetic workloads. *Proceedings of the SCS Multiconference on Distributed Simulation*, 22(1):23–28, Jan. 1990.

[10] R. M. Fujimoto. Performance measurements of distributed simulation strategies. *Transactions of the Society for Computer Simulation*, 6(2):89–132, Apr. 1989.

[11] R. M. Fujimoto. *Parallel and Distributed Simulation Systems*. Wiley Interscience, Jan. 2000.

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

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

[14] D. Jefferson and P. L. Reiher. Supercritical speedup. In A. H. Rutan, editor, *Proceedings of the 24th Annual Simulation Symposium*, pages 159–168. IEEE Computer Society Press, Apr. 1991.

[15] Y.-B. Lin. Parallelism analyzer for parallel discrete event simulation. *ACM Transactions on Modeling and Computer Simulation*, 2(3):239–264, July 1992.

[16] 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.

[17] M. Livny. A study of parallelism in distributed simulation. In *Proceedings 1985 SCS Multiconference on Distributed Simulation*, pages 94–98, Jan. 1985.

[18] E. J. Park, S. Eidenbenz, N. Santhi, G. Chapuis, and B. Settlemyer. Parameterized benchmarking of parallel discrete event simulation systems: Communication, computation, and memory. In *Proceedings of the 2015 Winter Simulation Conference (WSC '15)*, 2015.

[19] 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.

[20] R. Rong, J. Hao, and J. Liu. Performance study of a minimalistic simulator on XSEDE massively parallel systems. In *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment*. ACM, 2014.

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