2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation

Scalable Time Warp on Blue Gene Supercomputers

David W. Bauer Jr.
High Performance Technologies, Inc.
Reston, Virginia 20190
Email: dbauer@hpti.com

Christopher D. Carothers and Akintayo Holder
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, New York 12180
Email: {chrisc,holdea}@cs.rpi.edu

***Abstract*—In this paper we illustrate scalable parallel performance for the Time Warp synchronization protocol on the L and P variants of the IBM Blue Gene supercomputer. Scalable Time Warp performance for models that communicate a large percentage of the event population over the network has not been shown on more than a handful of processors. We present our design for a robust performing Time Warp simulator over a variety of communication loads, and extremely large processor counts – up to 131,072. For the PHOLD benchmark model using 65,536 processors, our Time Warp simulator produces a peak committed event-rate of 12.26 billion events per second at 10% remote events and 4 billion events per second at 100% remote events, the largest ever reported. Additionally, for the Tranmission Line Matrix (TLM) model which approximates Maxwell's equations for electromagnetic wave propagation, we report a committed event-rate in excess of 100 million on 5,000 processors with 200 million grid-LPs. The TLM model is particularly challenging given the bursty and cubic growth in event generation. Overall, these performance results indicate that scalable Time Warp performance is obtainable on high-processor counts over a wide variety of event scheduling behaviors and not limited to relatively low, non-bursty rates of off-processor communications.**

I. Introduction

Since its inception in 1985 [1], the Time Warp synchronization protocol for parallel discrete event simulation has been limited in its ability to scale to many processors and for models that communication a large amount of data between them. Known as *optimistic simulation*, the fundamental premise behind Time Warp is to only "synchronize" when absolutely necessary. A major feature of the Time Warp protocol is the rollback mechanism which corrects erroneous event processing by undoing the state of the model to correct for events that arrive out-of-order. The rollback mechanism has widely been viewed as the limiting factor in these systems because the number and depth of rollbacks cannot be bounded [2].

Until ten years ago, the general implementation approach for rollback mechanisms was to save the state of the model such that the entities, or logical processes (LPs), could be restored when errors in the *causality constraint* [3] occur, and a great deal of research has been performed investigating optimizations for state-saving[4], [5], [6], [7], [8], [9], [10], [11], [12].

In 1999, a second technique was proposed called *reverse computation* [13]. This technique restores the state of the simulation entities by computing the inverse operations for each event being rolled back. While this approach is not without its limitations (e.g., the irreversibility of floating point operations without loss of precision), it has allowed for much larger scale models because entity state is no longer preserved by the simulator, conserving large amounts of memory. Frequently these limitations can be overcome through high level inspection of the model, and code can be generated to compute the inverse operations less directly than at the instruction level [14], [15]. In addition to memory savings, this approach typically involves minimal overhead in the forward execution of events, minimizing the costs for supporting rollback in the common case – that is, events are processed in timestamp order and no causal error occurs.

Our contribution is demonstrable, scalable parallel performance of a Time Warp simulator employing reverse computation executing on the well-balanced hardware architecture of the IBM Blue Gene supercomputer. In particular, our results demonstrate strong-scaling up to 32,768 processors on a Blue Gene/L and strong-scaling up to 65,536 processors on a Blue Gene/P. This level of scaling is attributed to our core Time Warp message/network management algorithms that leverage the high-performance asynchronous message capabilities of the Blue Gene architecture. In the remainder of this paper, relevant data structures and algorithms used in our implementation are described in Section III. Our performance study using both the PHOLD benchmark and the Transmission Line Matrix application is described in Section IV. Previous research is reviewed in Section V and concluding remarks can be found in Section VI. We begin with an overview of the Blue Gene architecture.

II. The Blue Gene Architecture

The Blue Gene philosophy holds that more powerful processors are not the answer when it comes to winning the massively parallel scaling war [16]. Instead, the Blue Gene/L architecture balances the computing power of the processor against the data delivery speed of the network. This philosophy led designers to create smaller, lower power compute nodes comprising two 32-bit IBM PowerPCs running at only 700 MHz with a peak memory per node of 1.0 GB. Each Blue Gene rack is composed into 1,024 nodes consisting of 32 drawers with 32 nodes in each drawer. Additionally, there are specialized I/O nodes that perform all file I/O. Nominally there is one I/O node for every 32 compute nodes.

Interconnecting both drawers of nodes and racks are five specialized primary networks. The most relevant for Time Warp implementations are the point-to-point and the global collective networks. The point-to-point network is a 3-D torus consisting of 12 bi-directional links with a bandwidth of 175 MB/s each in the X, Y and Z directions. The global collective

1087-4097/09 $25.00 © 2009 IEEE
DOI 10.1109/PADS.2009.21

network enables data collection, reduction and redistribution to all nodes (or a subset) with a latency of 5 μs. As we will see in Section III, this collective network is critical to Time Warp's ability to efficiently compute Global Virtual Time (GVT) and re-claim memory. The interface for sending data over the Blue Gene network is MPI [16]

The Blue Gene/P is the successor system to the Blue Gene/L which can scale to 884,736 processors. Each compute node consists of four, 850 MHz PowerPC processors. The network design is very similar to the Blue Gene/L however, the point-to-point network has a 2.4x increase in bandwidth.

For this experimental study both a 32,768 processor Blue Gene/L and a 163,840 processor Blue Gene/P are used. The Blue Gene/L, named *fen* is located at the Rensselaer Computational Center for Nanotechnology Innovations (CCNI). The Blue Gene/P, named *Intrepid* is located at Argonne Leadership Computing Facility at Argonne National Laboratory.

Finally, we note that the IBM XLC C compiler was used for all the results in this paper. We were able to take full advantage of the compiler's peak optimization level as well as architecture specific settings. Our specific compiler options where: `-O5 -qarch=440d -qtune=440` for Blue Gene/L and `-O5 -qarch=450d -qtune=450` for Blue Gene/P.

III. Data Structures & Algorithms

The Time Warp protocol is inherently distributed in the construction of data structures, and the application of algorithms. The single exception to this statement is the *global virtual time* (GVT) algorithm that must compute the lowest timestamp among all unprocessed events in the system. In this section we outline our choices for data structures and algorithms that implement the Time Warp protocol in a stable and robust system for execution on high processor count supercomputers. The code for this implementation is based on ROSS: Rensselaer's Optimistic Simulation System [17], [18].

A. Events & Hashtables

Two types of events are possible in a Time Warp system: *positive* and *canceled*. For the sending processor, both types are represented in memory as a single event with a bit that signifies positive when set to zero, and canceled if one. When the sending processor sends a positive event, it creates a unique event id and sends the event using `MPI_Isend`. For cancelation events, the cancel bit is set to one, or true, and the event is re-sent without modifying the original unique id.

For the receiving processor, local event memory must be used to read events in from the network and the event bitfield must be inspected to determine if the event is positive or canceled. Upon receipt of a positive event, a pointer to the event is also stored into a hashtable. If a cancelation event arrives at a later time, the local copy of the event can then be retrieved from the hashtable using the unique id stored in the event.

Each processor maintains an array of hashtables [19], one for each processor in the system, and with a starting size of seven (7) to minimize memory requirements. The hash implementation is a straightforward array-based hashtable with quadratic probing (default load factor: $\lambda = 0.50$). Upon insertion of events, when the load average is exceeded (e.g., 50% of 7 would be the fourth insertion) the next prime number above the current size is identified and the elements are rehashed into the new hashtable. Please note that only the hashtable for the sending processor is resized, rather than all, and the sizes become model-dependent.

The average runtimes for searching a hashtable with quadratic probing are:

$$\frac{1}{(1-\lambda)} \tag{1}$$

for an unsuccessful search, and

$$\frac{1}{\lambda}\log\frac{1}{(1-\lambda)} \tag{2}$$

for a successful search. Using $\lambda = 0.5$, the average number of probes is then 2.0 for an unsuccessful search, and 1.4 for a successful search. The timings for unsuccessful searching correspond to inserting positive events, and successful searching correspond to deleting canceled or fossil collected events.

B. Flow Control

For any model that generates even a small amount of network traffic, flow control quickly becomes a necessity. In our implementation, flow control was performed on the sender's side by maintaining a FIFO queue for events that exceeded the capacity of the network hardware. Flow control on the receiver's side is performed to control the amount of time spent receiving events in the simulator. In particular, if too much time is spent receiving events, we may either exhaust the free pool of events available, or the processor may unnecessarily fall behind other processors in the system and create rollbacks.

The simulator defines input parameters for the size of the send and receive buffers and these values are used to initialize the size of the `MPI_Request` and `MPI_Status` arrays. We found that static sizes are sufficient and that a adaptive flow control mechanism as described in [20] was not necessary for a message-passing implementation of Time Warp.

During the call to `MPI_Testsome` these arrays are populated with event data, and the status of the network operation performed. When complete, the total number of events ready for processing is returned. After the events are processed, we iterate through the arrays, collapsing them so that fragmentation does not occur.

C. Asynchronous Message Passing

In our implementation, we instantiate a single MPI process on each processor. Events are communicated between processors using the asynchronous message passing portion of the MPI library through the routines `MPI_Isend` and `MPI_Irecv`. The algorithm for both sending and receiving events is then a straightforward two stage process for asynchronous message passing. The open question is, how do we implement this process efficiently in a Time Warp system?

When sending an event, the event is initially sent by calling `MPI_Isend` and the simulator returns to processing. At a later time we test that the send has completed and check that the event has not been canceled. Upon completion, if the event has not been canceled, it can later be reclaimed when the

event that caused this event to be sent is reclaimed. Otherwise, the event is marked as canceled and resent to the destination processor.

When receiving an event, the event is initially received by calling `MPI_Irecv` and the simulator returns to processing. At a later time we test that the receive has completed, and if the event has not been canceled, the event is stored in the local processors inbound event queue. Otherwise, a local copy of the event is retrieved from the hashtable and placed into the local processors canceled event queue. Finally, the received cancelation event is returned to the free event queue.

Algorithm 1 Implementation the Time Warp event scheduler & processing loop. The key parameters are `GVT_interval` and `batch_size`.

```
1:  while true do
2:    process network queues
3:    if !GVT_interval- - then
4:      start GVT computation?
5:    end if
6:    process inbound event queue
7:    process canceled event queue
8:    if GVT computation started then
9:      compute GVT
10:     reset GVT_interval
11:   end if
12:   if simulation end time reached then
13:     break
14:   end if
15:   process batch_size events
16: end while
```

Algorithm 1 illustrates our implementation of asynchronous message passing for the main Time Warp scheduler and event processing loop. The first action in the scheduler is to process network event sends and receives. The frequency at which this polling is done is directly affected by the number of events processed during the batch execution. In order to poll frequently, a small `batch_size` can be selected.

Algorithm 2 Processing of network queues.

```
1: changed ⇐ 1
2: while changed do
3:   changed |= test_q(posted_recvs, recv_finish)
4:   changed |= test_q(posted_sends, send_finish)
5:   changed |= recv_begin(...)
6:   changed |= send_begin(...)
7: end while
```

Algorithm 2 expands line 2 of Algorithm 1. The function `test_q` takes as a parameter a pointer to an array of outstanding events for either sending or receiving, and determines if any events have completed. For each completed event, the appropriate function is called, either `recv_finish` or `send_finish`. After attempting to complete outstanding sends / receives, the algorithm then attempts to start additional event sends and receives that have been queued by the flow control mechanism. Completing outstanding network operations takes precedence over new operations, and reading events takes precedence over sending events. The order of operations is important because the event pool is statically allocated during initialization of the simulation, and the desired bias is to drain the network as quickly as possible.

Algorithm 3 Processing network send buffers during batch execution.

```
1: changed ⇐ 1
2: while changed do
3:   changed |= test_q(posted_sends, send_finish)
4:   changed |= send_begin(...)
5: end while
```

The second time the network library is invoked is during batch execution when an event is sent remotely over the network to another processor. Algorithm 3 illustrates the process for sending events over the network. Initially, we tried using Algorithm 2 for this purpose, but found that polling the network this frequently was too expensive. While we initially wanted to be as aggressive as possible in pulling data from the network in an effort to minimize straggler events, we determined through experimentation that the overhead of the `MPI_Testsome` function was too high and leads to a decrease in the simulation performance.

D. Global Virtual Time

Efficient GVT computation is critical to the performance of Time Warp because it allows for the reclamation of events with a timestamp less than the computed GVT value. Our implementation is a variation of Mattern's GVT algorithm where events are 'colored' so that they can be accounted for during GVT computation [21]. Mattern describes an algorithm that accounts for all unprocessed events in a system by coloring events white that are sent over the network prior to a GVT computation, red during a GVT computation, and back to white after a GVT computation has concluded. These stages are synchronized across multiple processors through the construction of a 'consistent cut', usually relying on one or more rounds of token passing between processors.

A GVT computation is initiated each time the main event scheduler loop (see Algorithm 1) executes `GVT_interval` iterations. Typically values for `GVT_interval` range from 512 to 2048 and `GVT_interval` $*$ `batch_size` number of events are processed between successive GVT computations. These two parameters can have a significant impact on model performance as they help to indirectly "throttle" overly optimistic execution by either increasing GVT frequency or increased polling frequency for off-processing messages.

Once a GVT computation is instantiated, the first step is to account for all white events in the system. Our algorithm for the Blue Gene accounts for all white events in the system by computing a reduction on the number of white events sent and received by each processor, and calls the `MPI_Allreduce` function for this computation, similar to [18].

Algorithm 4 illustrates our GVT implementation using MPICH. The loop invariant must eventually become zero,

Algorithm 4 Variation of Mattern's GVT algorithm for Blue Gene.

1: $total\_white \Leftarrow 1$
2: **while** total_white **do**
3: read events from network
4: $local\_white \Leftarrow nwhite\_sent - nwhite\_recv$
5: MPI_Allreduce(&$local\_white$, &$total\_white$, 1, MPI_LONG_LONG, MPI_SUM, MPI_COMM_WORLD)
6: $count++$
7: **end while**
8: MPI_Allreduce(&lvt, &gvt, 1, MPI_DOUBLE, MPI_MIN, MPI_COMM_WORLD)

as the reduction of the variable `local_white` across all processors should yield a result of `total_white` = 0. Once all white events have been accounted for in the system, the GVT computation can proceed by reducing the minimum timestamp among all unprocessed events in the system.

Because the point-to-point and global collective networks are separate, nearly independent "data channels" on the Blue Gene and other architectures, it is possible for a point-to-point message to be sent before a reduction operation starts yet have the reduction complete first. Thus, a number of "round" through the GVT computation loop maybe necessary. As a network performance metric, the `count` variable is used to compute the average number of times through the loop per GVT computation. We have observed that the average number of read attempts required per GVT computation is 1 on Blue Gene/L, and 2 on the Blue Gene/P. On commodity clusters consisting of thousands of compute nodes, we have observed significantly higher averages (e.g., > 1000 on clusters containing 1,000s of processors). While the network architecture of the Blue Gene significantly improves the performance of this GVT algorithm in Time Warp, we have been unable to ascertain why the Blue Gene/P machine requires a second reduction attempt. Our hypothesis is that the shared memory bus of the Blue Gene/P may be less efficient in our implementation than directly using the Blue Gene network however more experimentation is needed.

E. Floating Point Timestamps and Memory Alignment

The importance of proper memory alignment of floating point timestamp values has been well documented for good performance on the Blue Gene [22]. In order to obtain good floating point performance all `double` floating point variables must be aligned on an 8 byte boundary. We discovered that while the Blue Gene/L is very tolerant of unaligned `double` timestamp variables, the Blue Gene/P is not. If an application encounters a single misaligned `double`, the Blue Gene/P generates a segmentation fault resulting in a core dump file for each processor.

To solve this alignment problem, we constructed our own dynamic memory subsystem to ensures each and every data structure starts on an 8 byte boundary address. Additionally, we manually re-ordered all data structure variables such that any `double` typed structure element occurred on an 8 byte offset within the defined structure type.

IV. Performance Study

We begin our performance investigation with a tuning study for best parameter settings of the simulator on the smaller, 32,768 processor Blue Gene/L. We then use those settings to perform measurements on the larger 131,072 processor Blue Gene/P. Because of the low noise of the Blue Gene architecture as show in [23], run-time variations are typically less than 1% and so replicated experiments are not required. Additionally, because of the limited processor allocation on these rare resources, we did not vary the random number seeds used because the long period and high quality of L'Ecuyer's Combined Linear Congruential Random number generator incurs little variance when different seeds are used (e.g., $<$ 0.1%) [14].

For the performance study we use the de-facto standard PDES benchmark, PHOLD, a derivative of the HOLD model [24], extended for parallel discrete event simulation in [25] and reverse computation in [13]. The PHOLD model generates a synthetic workload over a range of parameters that allow performance assessments based on application characteristics, rather than an intuitive understanding of a specific application. For this study we have chosen what are considered to be the most difficult settings: fine granularity event processing (e.g., none), and an inter-event offset based on an exponential distribution with a mean of 1.0, and destination LPs chosen randomly over the entire set of LPs. Configured in this way, the PHOLD benchmark rapidly generates a large number of remote events, close together in the simulation timeline with no built-in lookahead.

We observe that *strong scaling* the PHOLD model is more difficult than weak scaling, and choose the total number of LPs to be 1,048,576 (1024$*$1024). For the majority of cases the number of kernel processes (KPs) was chosen to be 32 per processor. KPs were first introduced as a construct for aggregating rollbacks and fossil collection across multiple LPs mapped to a single processor [17]. Each LP initially schedules 10 events into the simulator for a total event population at any time of approximately 10 million events. This configuration is identical to the previously published experimental study detailed in [18]. The model computes over 50 billion events with an end time of 5000.0.

As a second model, performance results are reported for the Tranmission Line Matrix (TLM) model [26] and [27] which approximates Maxwell's equations for electromagnetic wave propagation. The configuration and details of this model is described in Section IV-D.

A. Metrics

For the performance results we use three primary metrics: *committed event-rate, remote events percentage* and *rollback efficiency*. The committed event-rate measures the number of events processed per second in the simulator *excluding rolled back events*. These are the same events that would occur in a sequential execution of the simulation. The remote events percentage measures the number of events transmitted over the network over of the entire runtime of the model as a percentage of the total number of events processed, not including rollback

Algorithm 5 Algorithm for selecting a destination LP in the PHOLD model according to a user supplied remote event percentage.

```
1:  NLP ⇐ total LPs in simulation
2:  offset_lpid = my rank * nlp_per_pe
3:  dest_lp ⇐ -1
4:  if RANDOM_DOUBLE(0..1) ≤ remote event percentage
    then
5:     dest_lp ⇐ RANDOM_INT(0, NLP)
6:     dest_lp + = offset_lpid
7:     if dest_lp ≥ NLP then
8:        dest_lp − = NLP
9:     end if
10: else
11:    dest_lp = cur_lp
12: end if
```

events. In addition, the model is constructed such that the remote events percentage can be prescribed, as shown in Algorithm 5. Lines 6–9 serve to ensure the destination LP was not randomly chosen to be an LP mapped to the current processor.

Finally, we redefine the rollback efficiency of the simulator by the equation:

$$1.0 - \frac{E_{RB}}{E_{Net}} \tag{3}$$

where E_{RB} is the total number of events rolled back and E_{Net} refers to the number of events processed in the sequential case. This calculation yields a more intuitive representation of the rollback efficiency by constructing a ratio based on the number of events rolled back compared to the total events processed in a sequential computation. For example, if $E_{RB} = E_{Net}$, then this formula computes an efficiency of zero, rather than 50%, which misleads us into believing that the simulator had *some* real efficiency. Our equation also allows for the efficiency to be negative when more events are rolled back than are executed sequentially. For example, if twice the number of net events are rolled back, then the efficiency is -100.0%. Finally, we find that this revised measure of efficiency better correlates to actual speedup performance.

B. Performance: Blue Gene/L

We start our performance analysis on the Blue Gene/L using 8,192 processors and by testing the effects of various remote events percentages on the simulator parameters: `batch_size`, `GVT_interval`, number of KPs, and the size of the send and receive buffers in the flow control mechanism. The goal is to experimentally determine the best settings for the simulator before moving to the larger Blue Gene/P. For these examinations, default simulator settings were a `batch_size` of 4 events, a `GVT_interval` of 2048 loop iterations, 16 kernel processes [17], optimistic memory of 32,768 events, and send / receive buffer sizes of 150,000 events.

The `batch_size` parameter varies the number of events processed during the main Time Warp scheduler loop. Figure 1 varies the `batch_size` from 1 to 8 before performance begins to flatten out. For the 25, 50, 75 and 100% remote events, the best `batch_size` setting is clearly 2. For the 10% remote events case, the `batch_size` is commonly used to reduce the amount of network traffic, the frequency of polling the network becomes significant and performance is better with larger `batch_size` values. Larger values allow more events to be sent through the network and the polling overhead is reduced.

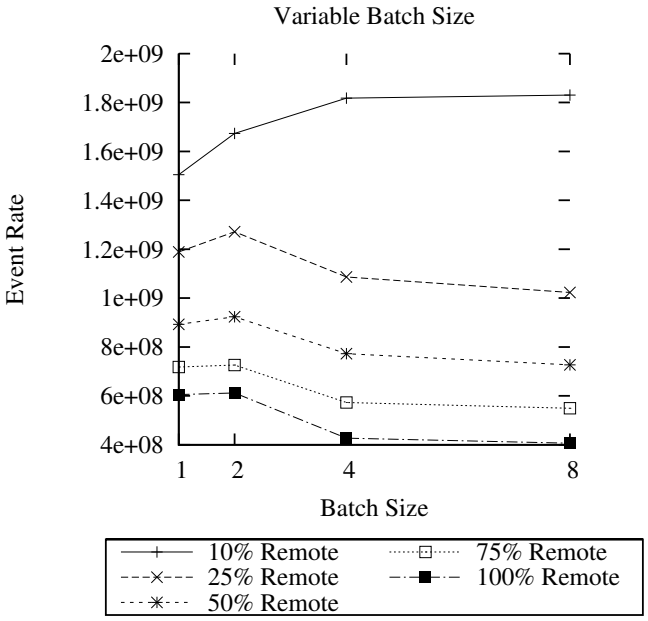

Fig. 1. PHOLD event-rate on 8192 processors: Varying `batch_size`. For large remote event percentages, the best `batch_size` setting is 2. Event-rate is measured interms of committed events per second and does not include any rolled back events.

The `GVT_interval` parameter varies the number of times the main scheduler loop is executed between GVT computations. Figure 2 varies the `GVT_interval` values from 64 upto 3K. Again, the best setting for the 25% to 100% remote events cases is the same, 512. And again, the 10% remote events is better when larger `GVT_interval` values are used. The frequency of fossil collection is tied to the `GVT_interval`. When fossil collection is delayed and the remote event percentage is high, the size of the hashtables grow and rehash more frequently, adding a great deal of overhead to the simulation. When the remote event percentage is low, this overhead is reduced, allowing for less frequent GVT computation and fossil collection. Conversely, computing GVT frequently for high remote event percentage minimizes the overhead of the hashtables, and for low event-rates unnecessarily increases the overhead associated with computing GVT yielding little to no overall performance benefit.

Kernel processes were introduced to improve the scaling of Time Warp by aggregating rollback and fossil collection for large-scale simulations. Varying the number of kernel processes in the simulator directly impacts the rollback mech-

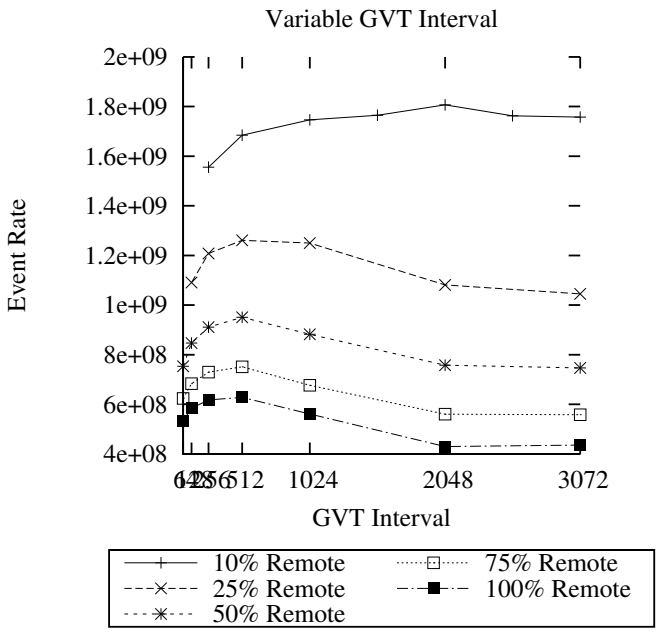

Fig. 2. PHOLD event-rate on 8192 processors: Varying the `GVT_interval`. The best settings for large remote event percentages is 512, and 2048 for smaller rates. Event-rate is measured interms of committed events per second and does not include any rolled back events.

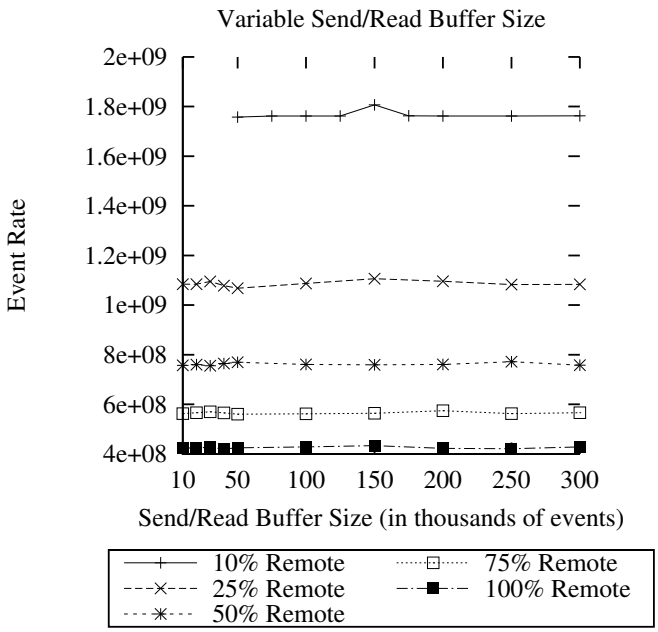

Fig. 4. PHOLD event-rate on 8192 processors: Varying the send/read buffer size from small sizes (10-50K at an interval of 10K) and large sizes from 50K to 300K, any of these settings work well on the Blue Gene. This indicates extremely robust performance of the network module. Event-rate is measured interms of committed events per second and does not include any rolled back events.

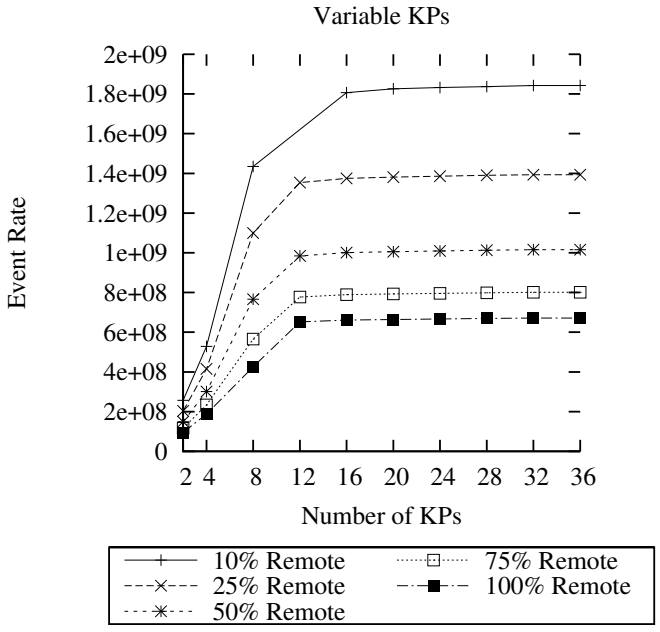

Fig. 3. PHOLD event-rate on 8192 processors: Variable number of kernel processes (KPs). The best setting is 16 KPs per processor. Increasing KPs/processors beyond 16 does not improve performance. Event-rate is measured interms of committed events per second and does not include any rolled back events.

anisms, as all processed events are preserved within the KPs until fossil collected. When too few KPs are used, rollbacks may occur unnecessarily as a consequence of rolling back the entire KP, rather than individual LPs. When the number of KPs is too large, the overhead associated with fossil collection becomes large. Typically a small number of KPs are required for large-scale simulations containing millions of LPs. Figure 3 illustrates the best KP configurations for various remote event percentage.

When the send and receive buffers were measured initially for large sizes (50–300K), the event-rate remained relatively unchanged, as shown in Figure 4. Subsequent testing in lower ranges (10–50K at an interval of 10K) revealed similar performance. These results indicate that the simulation network module is extremely robust. In addition, the well-balanced "low-noise" hardware architecture of the Blue Gene/L contributed to the stable performance.

Finally, we illustrate the scalability of our Time Warp implementation on the Blue Gene/L for a variety of processor configurations. Figure 5 shows super-linear performance for each remote event percentage tested on the Blue Gene/L. The linear scalability is due to the efficient implementation of the Time Warp protocol, while the super-linear performance is due to the model fitting more and more into the L3 cache as the number of processors is increased.

C. Performance: Blue Gene/P

Figures 6 and 7 show the scalability and efficiency for PHOLD on the Blue Gene/P. Because of our limited allocation, very few experiments at processor counts less than 65,536 were run. We believe that prior Blue Gene/L experiments

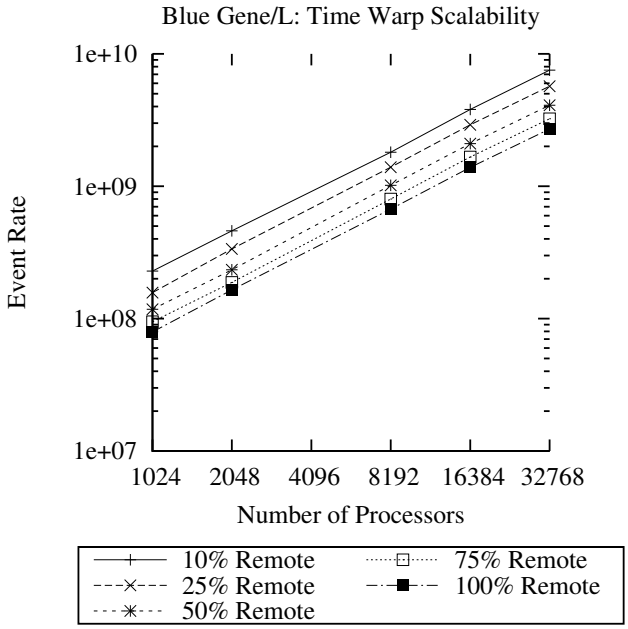

Fig. 5. Scalability of the Time Warp protocol for a variety of remote event percentages on multiple processor configurations. Event-rate is measured interms of committed events per second and does not include any rolled back events.

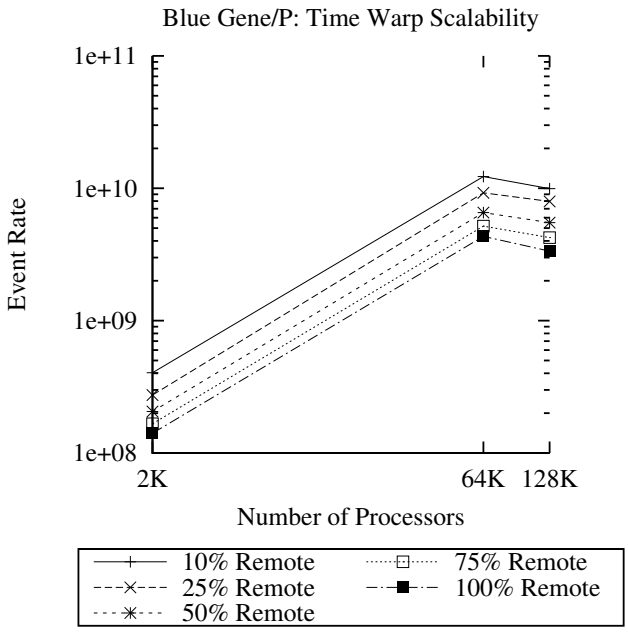

Fig. 6. Time Warp scalability on Blue Gene/P for PHOLD ranging from 2,048 (2K) to 131,072 (128K) processors. Event-rate is measured interms of committed events per second and does not include any rolled back events.

adequately covers those processor count ranges. Instead, we focused a majority of our runs on 65,536 and 131,072 processor configurations and compare that with results generated using only 2048 processors.

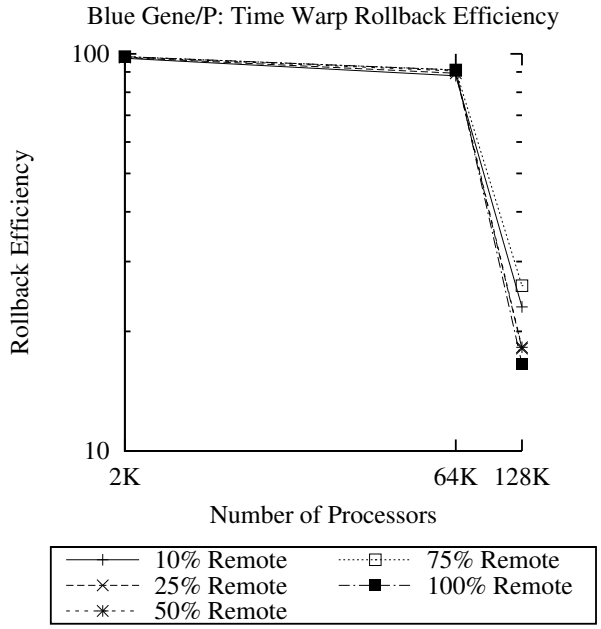

Fig. 7. Time Warp efficiency as a percentage on Blue Gene/P for PHOLD ranging from 2,048 (2K) to 131,072 (128K) processors.

The first phenomenon we encountered is that the Blue Gene/L's event-rate for 2,048 processors was significantly higher than Blue Gene/P's (410 million vs. 356 million) for the 10% remote events case. This is very odd given that the Blue Gene/P has a faster processor and higher bandwidth network. However, with additional experimentation, it was determined if Blue Gene/P was configured in `dual` mode to use only 2 of the 4 processors and increase the number of nodes from 512 to 1,024, our performance improved to 404 million. We believe there could be some memory bandwidth limitations on the Blue Gene/P that does not occur on the Blue Gene/L because of fewer processors per node, however more experimentation will be necessary to confirm this hypothesis.

Next, we observe scalable event-rate performance and $>$ 90% rollback efficiency for PHOLD across all remote event percentages upto 65,536 processors. At 65,536 processors, the lowest reported event-rate was 4 billion for the 100% remote events case and the highest reported event-rate was 12.26 billion for the 10% remote events case. However, when the 131,072 processor configuration was executed an avalanche of rollbacks occurred. We tried a number of Time Warp parameter changes, but the rollback-rate could not be improved. We believe here, that with only 8 LPs processors and 80 initial events per processor, that there is not enough local work per processor to sustain good optimistic event scheduling.

D. Performance: TLM on Blue Gene/L

The Transmission Line Matrix (TLM) model as described in [26] and [27] simulates microwave frequencies of light as a wave, based Christian Huygen's model of light. TLM approximates Maxwell's equations for electromagnetic wave propagation, with applications in radio wave communication networks.

This modeling approach is a mesh refinement technique that requires two main components: a discrete representation of the spatial environment modeled as a structured grid, and an algorithm for scattering and gathering of the electromagnetic field energies through the grid. The model is capable of capturing the primary effects of wave propagation, namely, *diffraction*, *reflection*, and *scattering*, each of which contributes to the multipath effect. An example wave transmission in shown in Figure 8. This model has the potential to provide highly accurate signal strengths for mobile ad-hoc networks and other wireless networking technologies as compared to traditional line-of-sight models.

Development of the TLM model first began in 2005 when the method was converted from a continuous description based on ordinary differential equations to the event-based paradigm [28]. In 2007, the performance of this model for the parallel discrete event paradigm was tested for a 100km^2 environment at a resolution of 100m and 754 radio transmitters [29] were used. Scalability was limited to 25 processors for this model, and speedups reported were about 50%. The limited scalability was primarily due to the fact that a large portion of the environment contained little or no work due to the large environment relative to the number of radios. This behavior is very different from the PHOLD workload. In PHOLD, the event population remains the same throughout the entire simulation run, making TLM a challenging model on which to yield good parallel performance.

Scaling TLM on the Blue Gene was complicated by the mapping of the grid cells (modeled by LPs) to physical processors (PEs). In our model, the terrain layer determines the bottom-most XY plane in the grid, and the model is decomposed in the X-axis. For an environment of size 100km^2 at a resolution of 100m, this yields 1,000 X-axis rows, limiting the number of processors used to 1,000. Increasing the size of the environment is unrealistic, as the mobility of the radios determines the size of the environment. Increasing the resolution of the environment allows us to scale up the size of the model and the benefit is more accuracy in the results generated. At a resolution of 10m, the same environment yields 10,000 X-axis rows, allowing us to utilize up to 10,000 processors. Scaling the model by increasing the resolution generates 100 million LPs for each Z-layer modeled, and for experiments we used two layers yielding a total of 200 million LPs.

We were able to collect results for 2,500, 5,000 and 10,000 processors. For our experiments, we selected the number of radio transmitters as 1, 10 or 100 per processor used in the simulation. That is, at 10,000 processors, the model has 1,000,000 radios. We consider the model to be an example of strong scaling, as the number of overall LPs in the model is largely determined by the grid size and resolution. Weak scaling the number of radios acts more to generate larger event populations, as more radio waves are being propagated at each timestep. Please note that a single wave starts as a single event in the spatial grid of LPs, and rapidly expanding out in all three dimensions simultaneously. Therefore, the event population grows according to the cubic volume of a sphere, deformed by the complex environment. A single transmission has the potential to generate millions of events, many with

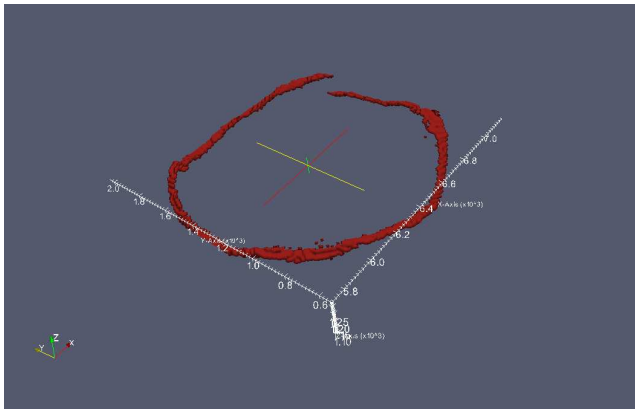

Fig. 8. Image of a single wave transmission in the TLM model. As the Z-axis is truncated, the wave becomes donut-shaped as it continues to propagate in the XY directions.

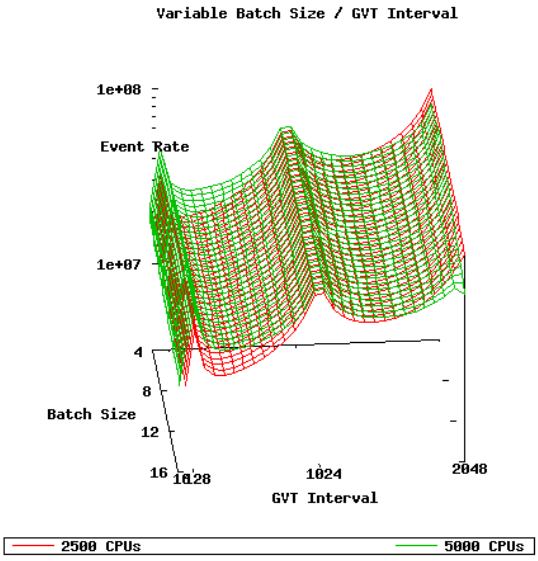

Fig. 9. The `GVT_interval` and `batch_size` surface plot. `GVT_interval` had the largest effect on the performance of the TLM model.

equal timestamps for a given instant in the simulation. The TLM model generates almost a worst case scenario for PDES in the growth of the event population and the high number of tied event timestamps. The remote event percentage for the 1 radio per processor cases was generally 17%, and 25-30% for the 10 and 100 radio per processor cases.

In order to determine the best settings for the TLM model, we measured a variety of `batch_size` and `GVT_interval` settings. For the TLM model, we determined our best settings were at 1,000 KPs per processor. Figure 9 illustrates the performance of the TLM model on the Blue Gene/L computing a full factorial over the `batch_size` and `GVT_interval` parameters. The `batch_size` had almost no effect on the performance, while larger GVT inter-

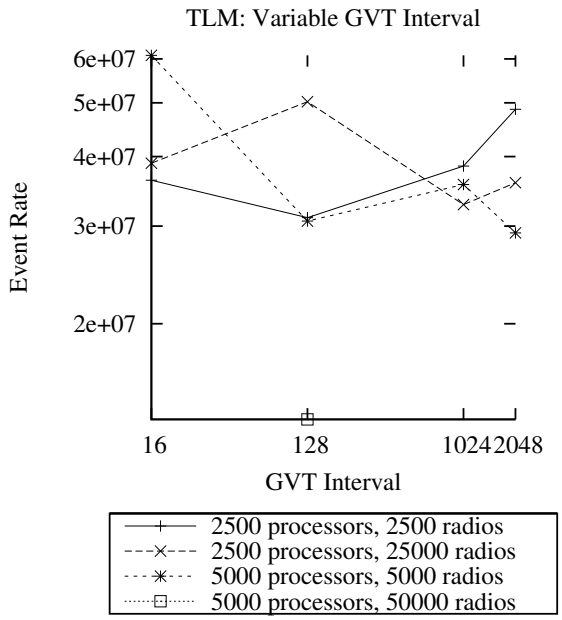

Fig. 10. TLM performance (event-rate) as a function of the `GVT_interval` for a varying number of processors and radios. Event-rate is measured interms of committed events per second and does not include any rolled back events.

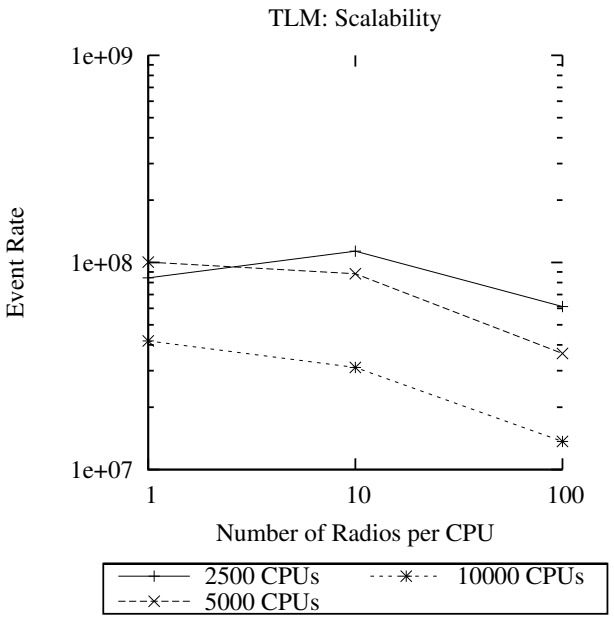

Fig. 11. Scalability of the TLM model for a range of radio transmitters per processor. Observe that the 5,000 processor configuration keeps up with the event-rate performance 2,500 processor case suggesting that weak scaling of the model is obtained.

vals improved the performance of the model from the worst case of 29 million events per second to over 100 million. Additionally, Figure 10 shows how the combination of GVT interval, number of processors and number of radios impacts the overall event-rate of the TLM model.

The scalability of the TLM model was measured for processor configurations: 2,500, 5,000 and 10,000, using a variety of radio transmitters. Figure 11 illustrates weak scaling of the TLM model since there is little difference in performance between 2,500 and 5000 processor cases, despite the doubling of the total number of radios. Because of the near cubic growth in event population as a radio wave propagates outward from its source, the 5000 processor cases has nearly 8x the event population that 2,500 processor case does. Consequently, each processor in the 5,000 processor case has upwards of 4 times the amount of work and event population per unit time which accounts for the loss of pure event-rate performance. In the 2,500 and 5,000 processor cases, the event-rate is above 100 million with a rollback efficiency of 2.6% for the 2,500 processor case and -4.32% for the 5,000 processor case indicating that the number of rollback events and number of net events processed (i.e., events that would occur in a sequential simulation) are nearly the same. Because the large number of radios, the 10,000 processor configuration yields a significant amount of work and even higher event population which explains why its overall event-rate is lower.

V. Related Work

There have been two previous investigation into the performance of discrete event simulation on Blue Gene supercomputers. The first study by Perumalla [22], presents PHOLD performance results for conservative, optimistic and mixed-mode PDES protocols on the Blue Gene/L. The peak optimistic PHOLD performance was 214 million events events per second on 8,192 processors for the 10% remote events case with built-in lookahead. The best reported event-rate was 530 million on 16,384 processors for purely conservative synchronization protocol.

The second study by Holder and Carothers [18] improves upon these results using their ROSS parallel simulator. Here, for PHOLD configured with 10% remote events and without lookahead, 853 million events per second on 16,384 Blue Gene/L processors is reported. A key difference between this implementation and that previous one is our use of non-blocking, asynchronous message passing primitives and the associated network data/buffer management layer that must be employed in order support models with high remote processor event sending rates. This resulted in a complete re-write of the core networking and event scheduling routines within the Time Warp/MPI implementation.

An additional PDES performance study on a 750 node Alpha server has also been investigated [30]. They were able to process an 228 million events per second on 1,024 Alpha processors. Their PHOLD model schedules events at one of the four nearest neighbors, which should exploit the quad processor SMP nodes while avoiding the Quadrics network switch. DSIM's GVT algorithm reserves processors as GVT managers. Typically, one manager is needed for every 128 processors. Because of the Blue Gene's built-in global reduction network, this software GVT approach is not required.

In the context of hardware acceleration of GVT algorithms, [31] demonstrates that hardware assisted, target-specific global reductions can dramatically improve parallel simulator per-

formance. More recently, [32] has shown the benefits of offloading the GVT computation to network interface cards.

VI. Conclusion

This paper demonstrates the capabilities of a new Time Warp system that leverages the high-performance asynchronous message passing capabilities of the Blue Gene family of supercomputers. We report peak PHOLD committed event-rates of 2 billion for 8,192 processors, 3.9 billion for 16,384, 7.6 billion for 32,768 and 12.26 billion for 65,536 processors. This represents a 14x improvement over previously published results. It is also demonstrates that when PHOLD is configured with very high remote processor sending percentages (e.g., $>$ 25%), Time Warp is able to maintain very good performance and scaling. On 65,536 Blue Gene/P processors, 4 billion events per second is reported for 100% remote events (e.g., every event processed was sent between two LPs located on different processors). This is only a 3x performance loss for 10x increase in remote events. Typically such high rates of off-processor communications results in much steeper losses in parallel simulator performance. Finally, it was demonstrated that a bursty event application, the Tranmission Line Matrix (TLM) model for electromagnetic wave propagation, was able to demonstrate weak scaling interms of number of radios from 2,500 to 10,000 processors. The largest previous processor configuration used for this model was limited to 25.

Overall, these performance results indicate that scalable Time Warp performance is obtainable on high-processor counts over a wide variety of event scheduling behaviors and not limited to relatively low, non-bursty rates of off-processor communications.

References

[1] D. R. Jefferson, "Virtual time," *ACM Trans. Program. Lang. Syst.*, vol. 7, no. 3, pp. 404–425, 1985.

[2] B. D. Lubachevsky, A. Shwartz, and A. Weiss, "An analysis of rollback-based simulation," *ACM Transactions on Modeling and Computer Simulation*, vol. 1, no. 2, pp. 154–193, 1991.

[3] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[4] D. West, "Optimizing time warp: lazy rollback and lazy re-evaluation," Master's thesis, University of Calgary, 1988.

[5] Y.-B. Lin, B. R. Preiss, W. M. Loucks, and E. D. Lazowska, "Selecting the checkpoint interval in time warp simulation," in *PADS '93: Proceedings of the seventh workshop on Parallel and distributed simulation*. New York, NY, USA: ACM Press, 1993, pp. 3–10.

[6] J. S. Steinman, "Incremental state saving in speedes using c++," in *WSC '93: Proceedings of the 25th conference on Winter simulation*. New York, NY, USA: ACM Press, 1993, pp. 687–696.

[7] J. Fleischmann and P. A. Wilsey, "Comparative analysis of periodic state saving techniques in time warp simulators," *SIGSIM Simul. Dig.*, vol. 25, no. 1, pp. 50–58, 1995.

[8] F. Gomes, "Optimizing incremental state-saving and restoration," Ph.D. dissertation, University of Calgary, 1996.

[9] F. Quaglia, "Fast-software-checkpointing in optimistic simulation: embedding state saving into the event routine instructions," in *PADS '99: Proceedings of the thirteenth workshop on Parallel and distributed simulation*. Washington, DC, USA: IEEE Computer Society, 1999, pp. 118–125.

[10] G. Chen and B. K. Szymanski, "Four types of lookback," in *PADS '03: Proceedings of the seventeenth workshop on Parallel and distributed simulation*. Washington, DC, USA: IEEE Computer Society, 2003, p. 3.

[11] L. Li and C. Tropper, "Event reconstruction in time warp," in *PADS '04: Proceedings of the eighteenth workshop on Parallel and distributed simulation*. New York, NY, USA: ACM Press, 2004, pp. 37–44.

[12] Y. Zeng, W. Cai, and S. J. Turner, "Batch based cancellation: a rollback optimal cancellation scheme in time warp simulations," in *PADS '04: Proceedings of the eighteenth workshop on Parallel and distributed simulation*. New York, NY, USA: ACM Press, 2004, pp. 78–86.

[13] C. D. Carothers, K. S. Perumalla, and R. M. Fujimoto, "Efficient optimistic parallel simulations using reverse computation," *ACM Trans. Model. Comput. Simul.*, vol. 9, no. 3, pp. 224–253, 1999.

[14] P. L'Ecuyer and T. H. Andres, "A random number generator based on the combination of four lcgs," *Math. Comput. Simul.*, vol. 44, no. 1, pp. 99–107, 1997.

[15] D. W. Bauer and E. H. Page, "An approach for incorporating rollback through perfectly reversible computation in a stream simulator," in *PADS '07: Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 171–178.

[16] N. R. Adiga and et al., "An Overview of the Blue Gene/L Supercomputer," in *Proceedings of the ACM/IEEE Conference on Supercomputing*, November 2002, pp. 1–22.

[17] C. D. Carothers, D. W. Bauer, and S. O. Pearce, "Ross: A high-performance, low memory, modular time warp system," *Journal of Parallel and Distributed Computing*, 2002.

[18] A. Holder and C. D. Carothers, "Analysis of time warp on a 32,768 processor ibm blue gene/l supercomputer," in *2008 Proceedings European Modeling and Simulation Symposium (EMSS)*, 2008.

[19] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2001.

[20] K. Panesar and R. M. Fujimoto, "Adaptive flow control in time warp," in *PADS '97: Proceedings of the 11th Workshop on Principles of Advanced and Distributed Simulation*. IEEE Computer Society, 1997.

[21] F. Mattern, "Efficient algorithms for distributed snapshots and global virtual time approximation," *Journal of Parallel and Distributed Computing*, vol. 18, no. 4, pp. 423–434, 1993.

[22] K. S. Perumalla, "Scaling time warp-based discrete event execution to 10^4 processors on a blue gene supercomputer," in *Proceedings of the 4^{th} international conference on Computing frontiers*, 2007, pp. 69–76.

[23] P. Beckman, K. Iskra, K. Yoshii, S. Coghlan, and A. Nataraj, "Benchmarking the Effects of Operating System Interference on Extreme-Scale Parallel Machines," *Cluster Comput.*, vol. 11, pp. 3–16, 2008.

[24] J. Vaucher and P. Duval, "A comparison of simulation event list algorithms," *Communications of the ACM*, vol. 18, no. 4, pp. 223–230, 1975.

[25] R. M. Fujimoto, "Performance of time warp under synthetic workloads," pp. 23–28, January 1990.

[26] P. Johns and R. Beurle, "Numerical solution of 2-dimensional scattering problems using a transmission-line matrix," *Proc. IEE*, vol. 118, no. 9, pp. 1203–1208, 1971.

[27] P. Johns, "The Solution of Inhomogeneous Waveguide Problems Using a Transmission-Line Matrix," *Microwave Theory and Techniques, IEEE Transactions on*, vol. 22, no. 3, pp. 209–215, 1974.

[28] J. Nutaro, "A discrete event method for wave simulation," *ACM Trans. Model. Comput. Simul.*, vol. 16, no. 2, pp. 174–195, 2006.

[29] D. Bauer Jr and E. Page, "Optimistic parallel discrete event simulation of the event-based transmission line matrix method," in *Winter Simulation Conference, 2007*, 2007, pp. 676–684.

[30] G. Chen and B. K. Szymanski, "Time quantum GVT: A scalable computation of the global virtual time in parallel discrete event simulations," *Scalable Computing: Practice and Experience: Scientific International Journal for Parallel and Distributed Computing*, pp. 425–446, 2007.

[31] C. Pancerella and P. F. Reynolds, "Disseminating critical target-specific synchronization information in parallel discrete event simulation," in *PADS '93: Proceedings of the 7th Workshop on Principles of Advanced and Distributed Simulation*. IEEE Computer Society, 1993, pp. 52–59.

[32] R. Noronha and N. B. Abu-Ghazaleh, "Using programm-able nics for time warp optimization," in *Proceedings of the International Parallel and Distributed Processing Symposium*, 2002.

Acknowledgments: This research was supported by NSF Contracts CNS #0435259 and #0133488. Additionally, we wish to thank David Jefferson for his help in our obtaining discretionary access to `Intrepid`, the Blue Gene/P at ALCF/ANL. We finally thank Ray Loy and Tisha Stacey from ACLF/ANL as well as Adam Todorski from the CCNI for their collective support and guidance on performance tuning.