Proceedings of the 2010 Winter Simulation Conference
B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, eds.

ON DECIDING BETWEEN CONSERVATIVE AND OPTIMISTIC APPROACHES ON MASSIVELY PARALLEL PLATFORMS

Christopher D. Carothers

Department of Computer Science
Rensselaer Polytechnic Institute
110 8th Street
Troy, New York 12180, USA

Kalyan S. Perumalla

Computer Sciences and Engineering Division
Oak Ridge National Laboratory
P.O. Box 2008 MS-6085
Oak Ridge, TN 37831-6085, USA

ABSTRACT

Over 5000 publications on parallel discrete event simulation (PDES) have appeared in the literature to date. Nevertheless, few articles have focused on empirical studies of PDES performance on large supercomputer-based systems. This gap is bridged here, by undertaking a parameterized performance study on thousands of processor cores of a Blue Gene supercomputing system. In contrast to theoretical insights from analytical studies, our study is based on actual implementation in software, incurring the actual messaging and computational overheads for both conservative and optimistic synchronization approaches of PDES. Complex and counter-intuitive effects are uncovered and analyzed, with different event timestamp distributions and available levels of concurrency in the synthetic benchmark models. The results are intended to provide guidance to the PDES community in terms of how the synchronization protocols behave at high processor core counts using a state-of-the-art supercomputing systems.

1 INTRODUCTION

A parallel discrete-event simulation (PDES) system consists of a collection of *logical processes* or LPs, each modeling a distinct component of the system being modeled (e.g., router in a physical communications network). LPs communicate by exchanging timestamped event messages (e.g., denoting the arrival of a new job at that server). The goal of PDES is to efficiently process all events in parallel in global timestamp order. Two well-established approaches towards this goal are broadly called conservative processing and optimistic processing.

The seminal parallel discrete-event processing approach, falling under the category of conservative processing, is the Null Message algorithm developed by Chandy and Misra (Chandy and Misra 1979) and Bryant (Bryant 1977) (also known as the CMB algorithm, based on the inventors' names). In this algorithm, each logical process sends a "null message" to its neighboring processes upon executing each event. The "null message" contains a timestamp T that serves as a "promise" that the sending process will not later send a message with timestamp smaller than T to the receiving process. At any process, if the next local event to be processed has a timestamp that is greater than any of the received "null message" events, that process must wait until it receives the next wave of "null messages" such that all "null message" timestamps are greater than the timestamp of the next event to be processed. It has been shown that this algorithm avoids deadlock provided that there is no cycle in which a message could traverse without incrementing its timestamp (i.e., timestamp increments must be non-zero).

Alternatives to the Null Message algorithm for conservative execution employ a "global synchronization" approach. Promiment examples arethe Bounded Lag algorithm (Lubachevsky, Shwartz, and Weiss 1991), Time Buckets (Steinman 1993), YAWNS (Dickens, Nicol, Reynolds, and Duva 1996), and more recently Composite Synchronization (Nicol and Liu 2002). In its simplest approach, each LP is allowed to process events between the most recently computed "lower bound timestamp" (LBTS), which is the smallest unprocessed event in the system, plus the global lookahead value among all LPs. When no more local events can be processed, all LPs must determine via a

reduction/barrier algorithm, such as (Pancerella and Reynolds 1993), the next LBTS, disseminate it, and commence processing events.

In contrast to conservative approaches, Time Warp is a well-known optimistic synchronization mechanism, developed by Jefferson and Sowizral (Jefferson 1985) used in the parallelization of discrete-event models. The Time Warp mechanism uses a detection-and-recovery protocol to detect causality errors and synchronize the computation. Here, anytime an LP determines that it has processed events out of timestamp order, it "rolls back" those events, and re-executes them in the correct order. Analogous to the conservative LBTS computations, Time Warp systems execute a Global Virtual Time (GVT) algorithm which determines the smallest unprocessed or partially processed event in the system. The GVT value is used to reclaim LP state and event memory which is retained for supporting the rollback operations.

It is interesting to note some relevant statistics of published research that can be uncovered by an on-line literature search. Using the phrase *"Time Warp, Optimistic Parallel Simulation"* in the Google Scholar search engine, approximately 1970 results are returned, as of this writing. Similarly, the phrase *"Conservative Parallel Discrete Event Simulation* returns over 5200 results. A large number of the articles are focused on demonstration of scaling results in one class of synchronization protocol or the other. However, fewer than 10 results to date correspond to articles in which systems using more than 1,000 processor cores are studied. The principal contribution of our present article is in providing an experimental performance study to provide guidance to the PDES community in terms of how the primary PDES algorithms behave at relatively high processor core counts on a state-of-the-art supercomputer system.

1.1 Related Work

In the rich and colorful history of PDES, one of the well-known analytical results (by proponents of Time Warp) appeared in 1990, in which Lipton and Mizell (Lipton and Mizell 1999) showed that Time Warp can, under certain circumstances, outperform the Chandy/Misra/Bryant algorithms by an arbitrary amount. Given N processors, Time Warp can outperform the Null Message algorithm by a factor of N, they further showed that the opposite is *not* true: the Null Message algorithm can only outperform Time Warp by a constant factor. The key insight into this result is the set of conditions in which Time Warp is able "uncover" N times more eligible events to process in parallel than the CMB approach. However, this analysis does assume that rollback costs are fixed, which may not always be the case. (Lubachevsky, Shwartz, and Weiss 1991) analyzed rolled-based parallel simulations and demonstrated the possibility of a dangerous "phase-transition" from a well-behaved execution to an unstable execution whereby rollbacks cascade without end. Here, an "echo" model is shown where the size of the rollback grows in time. From these early results, there have been a series of improvements in both algorithms which we briefly summarize here. For conservative protocols, there is: Bounded Lag (Lubachevsky, Shwartz, and Weiss 1991), YAWNS (Dickens, Nicol, Reynolds, and Duva 1996), and Critical Channel Traversing (Xiao, Unger, Simmonds, and Cleary 1999), and Composite Synchronization (Nicol and Liu 2002). For optimistic approaches, there is Periodic State Saving (Bellenot 1992), Incremental State Saving (Gomes 1996) and Reverse Computation (Carothers, Perumalla, and Fujimoto 1999) all which lower state-saving overheads. In terms of GVT/LBTS algorithms there are: distributed snapshots (Mattern 1993), hardware assisted reduction (Pancerella and Reynolds 1993), shared memory approaches (D'Souza et al. 1994, Fujimoto and Hybinette 1997, Gomes et al. 1995), and clock/cycle counter-based approach (Bauer et al. 2005). The LBTS/GVT algorithm of (Perumalla and Fujimoto 2001) provided a unified algorithm to encompass the cross product of $(reliabile, unreliable) \times (events, messages)$, and was eventually used in the μsik system to scale to hundreds of thousands of processors (Perumalla 2007).

However, in terms of results on modern supercomputer class systems (e.g. systems still in service and on the Top500 supercomputer list), there are relatively few results. The first PDES performance study to focus on the Blue Gene supercomputer platform is (Perumalla 2007). Here, PHOLD performance results for conservative, optimistic and mixed-mode PDES protocols on the Blue Gene/L are presented using μsik parallel discrete-event simulator. The next two Blue Gene/PDES performance studies (Holder and Carothers 2008, Bauer Jr., Carothers, and Holder 2009) demonstrates the performance of their Time Warp simulator, ROSS, being the first to demonstrate that event-rates from the 100's of millions of events-per-second to even billions of events-per-second are possible and scalability out to 65,536 processors are possible. Most recently, it has been demonstrated that is possible to execute epidemic outbreak models (Perumalla 2010b) using 65,536 Cray XT5 processors and in (Perumalla 2010a) that modeling massively parallel MPI programs (with multiple millions of virtual MPI ranks) is possible using over 216,000

CrayXT 5 cores using the $\mu\pi$. These are the largest core-counts any PDES model has efficiently executed on to date.

This performance study adds to these previous results by demonstrating how model lookahead and event timestamp distribution impacts conservative and optimistic performance and provides model developers some guidance on when and where each protocol might provide a performance advantage over the other on supercomputer systems.

1.2 Performance Study Approach

The merits and shortcomings of optimistic and conservative execution have been well studied in the past. Nevertheless, a comparative study on their performance on large parallel computing platforms has remained relatively unexplored in the literature. Here, we design a set of experiments to uncover some of the important factors underlying their performance effects when executed with various event workloads on increasingly larger numbers of processor cores.

A few, simplified insights on the relative performance of conservative and optimistic execution have been well known for a long time. For example, it is well known that low lookahead values can make conservative execution perform exceedingly poorly. Similarly, optimistic execution overheads may be high when compared to excellent performance of conservative execution with high lookahead values. However, relative performance in the intermediate range of the lookahead spectrum, and in relation to several other important parameters, is not well understood.

Moreover, the specific characteristics of the hardware can have a significant impact on the runtime efficiency of either synchronization method. For example, the very high speed of global reductions afforded by special hardware (as the collectives network in the Blue Gene machines) can be expected to make conservative execution stay competitive with optimistic execution even with low lookahead scenarios. The high speed of communication can also, similarly, help optimistic execution by helping to keep the working set small (countering the increased memory use for optimistic processing). The reduced working set can help optimistic execution stay within the cache size, to make the event cost sufficiently low to compete well with conservative execution even with large lookahead.

Also, the amount of blocked time spent in synchronization depends on the time the last processor joins the collective synchronization. Such an imbalance is in turn affected by timestamp distribution, the number of events, and the event granularity. Note that the imbalance is not in terms of the number of *available* events per processor, but in fact in the number of *processable* events within a given simulation time window.

The effectiveness of using special hardware support, such as the collectives network in the Blue Gene machines, may be reduced due to the staggered event timestamp distributions.

With strong scaling, increasing the number of processors has the effect of increasing the minimum timestamp in the priority queues across processors. This indirectly affects the relative goodness of a given lookahead value, since the lookahead becomes smaller relative to the simulation time window defined by the combined set of all minimum timestamps across all processors.

1.3 Study Outline

It is the net, empirical results of all the aforementioned effects that are quantitatively explored here. Specifically, the following factors are considered in our study:

Lookahead: Here, we experiment with increasing values of lookahead (LA). We defined four values for lookahead, to which we will refer as *little LA*, *small LA*, *medium LA*, and *large LA*. For the specific benchmarks used in the experiments, they are 0.01, 0.1, 0.5, and 0.9, respectively. Note that lookahead values must be viewed relative to event time distribution. It is not sensible to evaluate as an absolute value, but it is only important as a quantity relative to the minimum timestamp in the priority queues of each processor.

Event timestamp distribution: This has a direct effect on the amount of concurrency presented to the simulation engine. To evaluate its effect, event timestamp distribution across processors is varied across processors within any given simulation time window. Two distinct distributions are exercised, to which we will refer as *staggered* and *non-staggered*. These two workloads are described later in the benchmark descriptions.

Processor core count: The effect of increasing the number of processor cores in strong scaling experiments is evaluated in three processor configurations, namely, *small*, *medium*, *large*. For the experiments, we used 1,024 cores for the small configuration, 8,192 cores for medium, and 16,384 for the large configurations. Although supercomputers with even larger configurations are indeed in existence, 16,384 processor cores represents a sufficiently large configuration on which detailed comparative performance is to be explored in parallel discrete event simulation.

Performance effects: The effect on overall performance by the speed and cost of synchronization (global virtual time or lower bound on timestamp) computations is also instrumented and analyzed in the experiments. When a good amount of concurrency is inherently available within the model, then, the specific synchronization scheme used can directly determines the parallel runtime performance, and poor performance often is reflected in the observed synchronization cost.

The metrics of interest in the experiments include the following: (i) Aggregate event rate, (ii) Global synchronization frequency, (iii) Blocked time spent in synchronization, (iv) Rollback efficiency of event processing (optimistic only), and, finally, (v) Parallel speedup and overall efficiency.

2 IMPLEMENTATION

2.1 Hardware Platform

For this performance study, we use the 32,768 processor Blue Gene/L system. Currently, this system ranks #80 on the Top500 Supercomputer List (see `www.top500.org`). What makes the Blue Gene system unique relative to other supercomputer systems, like the Cray XT series, is that the Blue Gene/L architecture balances the computing power of the processor against the data delivery speed of the network (Adiga and et al. 2002). This design goal led designers to create smaller, lower power compute nodes comprising two 32-bit IBM PowerPCs running at only 700 MHz each, with a peak memory of 1.0 GB per node. Each Blue Gene rack is composed from 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 PDES 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 network enables data collection, reduction and redistribution to all nodes (or a subset) with a latency of 5 μs. As we will see in the performance results, this collective network is critical to efficiently computing GVT and LBTS. The interface for sending data over the Blue Gene network is MPI (Adiga and et al. 2002).

2.2 Simulation Software

Figure 2.2(a) shows the high-level algorithm for ROSS' implementation of Time Warp. The first action in the scheduler is to process network event communication (sends and receives). Events are sent and recieved using `MPI_Isend` and `MPI_Irecv` respectively. These are asynchronous routines in that message send requests and recieve requests are posted to the underlying message transport layer but not necessarily complete when either of these routines returns. The simulation engine must determine if these message requests have completed before reusing the buffers holding the message data. The frequency at which this polling is done is directly affected by the number of events processed during the batch of events executed. In order to poll frequently, a small `batch_size`, as low as 1, can be selected.

A GVT computation is initiated each time the main event scheduler loop (see below) 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 the GVT frequency or increasing the polling frequency for off-processor messages. Once a GVT computation is instantiated, the first step is to account for all events in the system. This is accomplished by computing a reduction on the difference in the number of events sent and received by each processor, and calls the `MPI_Allreduce` function for this computation. For a more detailed description of ROSS we refer the reader to (Bauer Jr., Carothers, and Holder 2009).,

The conservative event scheduler is shown in Figure 2.2(b). The core part of the scheduler is very similar to that of Time Warp. First, the network queues are read for any new incoming messsages followed by inqueuing those new events in the priority queue. Next, a check is made to ensure that the next event to be processed is earlier than *LBTS* + *Lookahead*; otherwise, an LBTS computation is initiated and completed. If the computed LBTS is beyond the end time, then the simulation terminates, otherwise it enters the "batch" processing loop and stays there for `batch_size` events unless one of them falls outside the *LBTS* + *Lookahead* time window. The same `MPI_allreduce` reduction algorithm is used to compute LBTS as GVT.

```
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
```

(a) Optimistic

```
1: while true do
2:    process network queues
3:    process inbound event queue
4:    if smallest event >= LBTS + Lookahead then
5:       compute new LBTS
6:    end if
7:    if simulation end time reached then
8:       break
9:    end if
10:   process batch_size events suject to event.ts <
      LBTS + Looahead
11: end while
```

(b) Conservative

Figure 1: (a) Implementation of the Time Warp event scheduler loop. The key parameters are `GVT_interval` and `batch_size`. (b) Implementation of the conservative event scheduler loop. The key parameters are `Lookahead`.

2.3 Benchmarks

The benchmark used in our performance study is PHOLD. Here, PHOLD is configured with 10% remote messages with *any-to-any* inter-LP event communication schedule, with no specific neighborhood bias. A total of 1,048,576 LPs are used where each LP has 10 initial start events. This configuration is consistent with previous performance studies (Perumalla 2007, Bauer Jr., Carothers, and Holder 2009). Lookahead for PHOLD is configured to be one of `0.01`, `0.01`, `0.25`, `0.5`, and `0.90` but Time Warp is not configured to leverage the lookahead directly, but naturally gets exploited by the GVT computation due to event timestamps being later in time. From this the mean of the exponential distribution is set to $1 - Lookahead$. For optimistic event processing, reverse computation is used (Carothers, Perumalla, and Fujimoto 1999).

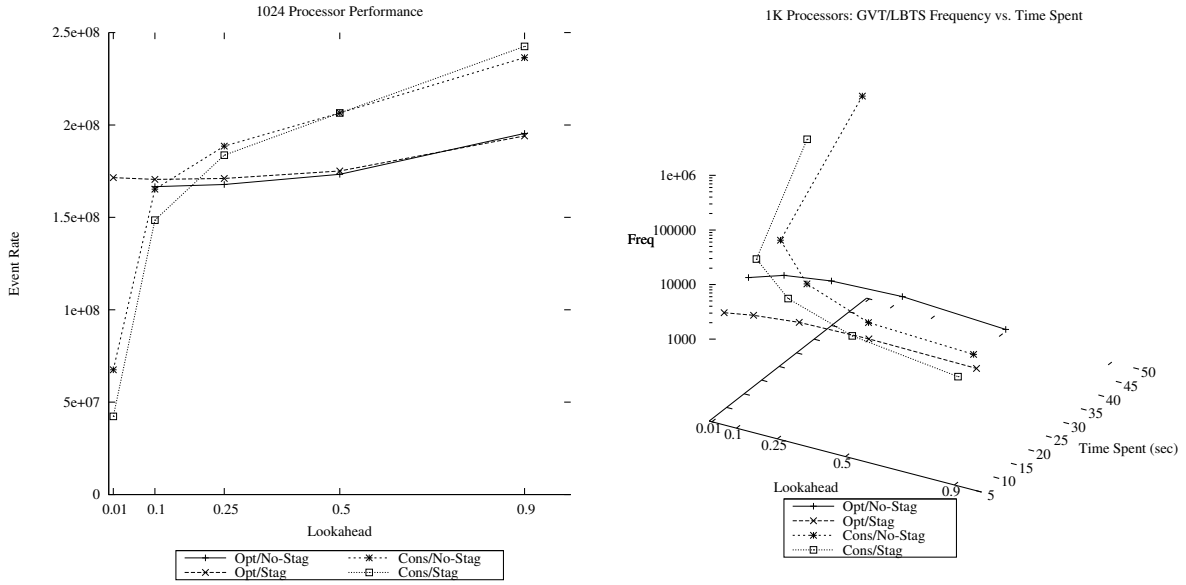
To properly exercise the potential variation of timestamp distribution across processors, we introduce a new parameter into PHOLD, we call "stagger". Here, event timestamps are drawn similar to the non-staggered (normal) case, except for one variation, which is that the initial timestamps (for all the PHOLD message population) are staggered in time across processors. The staggering is achieved by adding a varying amount of initial delay to the event timestamps generated in the initialization routine. This has the effect of varying the event timestamp distribution across processors (more precisely, LPs) within any given simulation-time window. In the non-staggered (normal) case, event timestamps are computed as $TS = Lookahead + EXP(1 - Lookahead)$ as previously indicated.

3 EXPERIMENTAL RESULTS

3.1 Observed Performance - 1K Cores

Figure 2(a) shows the observed event rates on 1,024 cores. As expected, conservative execution suffers poor performance with low lookahead values, while optimistic execution delivers excellent event rates even with the low lookahead setting. This is due to the fact that conservative execution must synchronize very frequently across all processors. The fact that synchronization cost is the underlying reason for poor performance is evident from the synchronization statistics plotted in Figure 2(b). With low lookahead, conservative execution results in over one million GVT computations, consuming majority of overall execution time in the time spent in GVT computations.

The poor performance of conservative execution with low lookahead, and the excellent performance of optimistic execution with the same lookahead, are both expected results. However, a remarkable observation is that the conservative execution recovers much of the performance loss even with a minor increase in lookahead. It is seen that, while the aggregate event rate is a low 50 million events per second with the low lookahead 0.01, it leaps to a



(a) Event rates

(b) GVT/LBTS frequency and timing

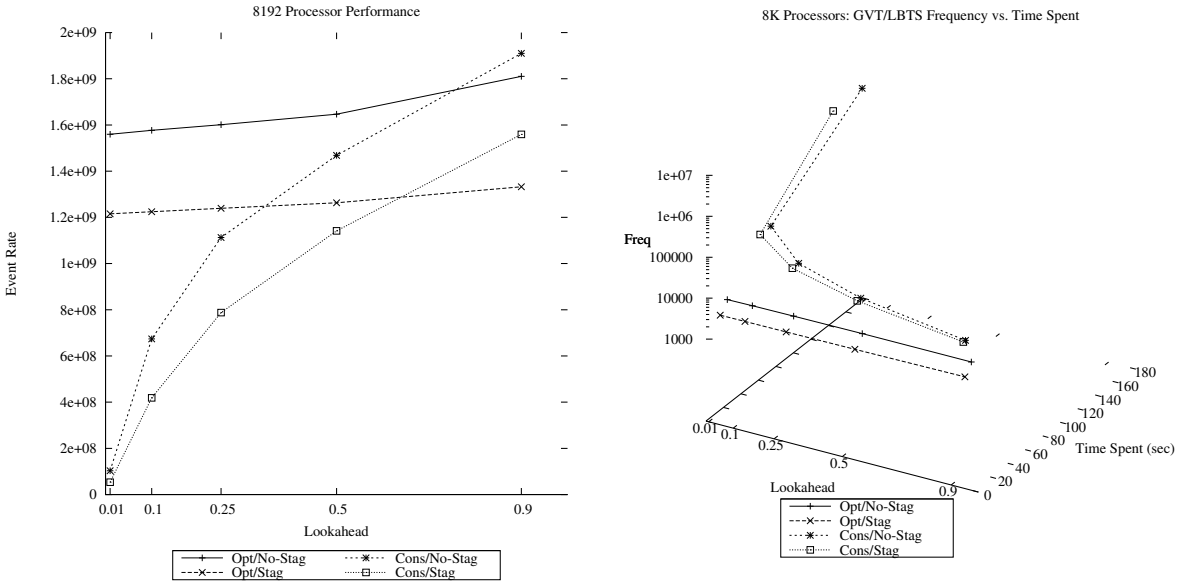
Figure 2: 1024 processor performance as a function of lookahead, and staggered vs. non-staggered variants of PHOLD models, with conservative and optimistic synchronization.

high event rate even with small lookahead of 0.1, making it competitive with optimistic execution. Also interesting is the fact that conservative execution continues to deliver improvements in event rate with increasing lookahead values, while optimistic execution does not improve its performance equally faster with increasing lookahead values. This can be explained by noting the time spent by optimistic execution on GVT computations with increasing lookahead. With increased lookahead, event timestamps are scattered a bit further in simulation time, making it possible for optimistic execution to be ill balanced (in simulation time) across processors. This ill balance makes the slower processor arrive at the GVT computation later than all others, thus making every processor incur the cost of blocked time. The increase in the total time spent in GVT computation (even while the *frequency* of GVT computation remains relatively constant) is evident in Figure 2(b), showing the increasing blocked time *per GVT computation* in optimistic execution. This observation is further supported by the fact that rollback cost does not contribute to this overhead: Figure 5 shows that the cost of rollbacks is negligible for 1K processor cores, giving close to 100% efficiency of event computation.

Note that optimistic execution also improves its performance with increasing lookahead (e.g., improving the event rate when moving from lookahead of 0.5 to 0.9). However, with large lookahead, the combination of the GVT computation overhead and other minor rollback-support cost contributes to a slightly lower (20%) performance of optimistic execution compared to conservative execution. With larger number of processors, however, we see (later in this section) that conservative execution fails to improve upon optimistic execution even with large lookahead.

The effect of staggered vs. non-staggered execution is observed to be insignificant in the 1K processor configuration. For conservative execution, this can be attributed to the fact that: (a) with low lookahead, both staggered and non-staggered executions offer little concurrency, resulting in very poor performance in both cases, and (b) with larger lookahead values, GVT is computed sufficiently fast to be able to exploit even small amounts of concurrency (only a few events per GVT per processor) enabled by the lookahead for the non-staggered case. Similarly, for optimistic execution, the amount of concurrency (in terms of the number of events available to safely process between two GVT computations) is sufficiently large even with staggered workload, making the benefits from higher concurrency in the non-staggered case insignificant compared to the staggered case.

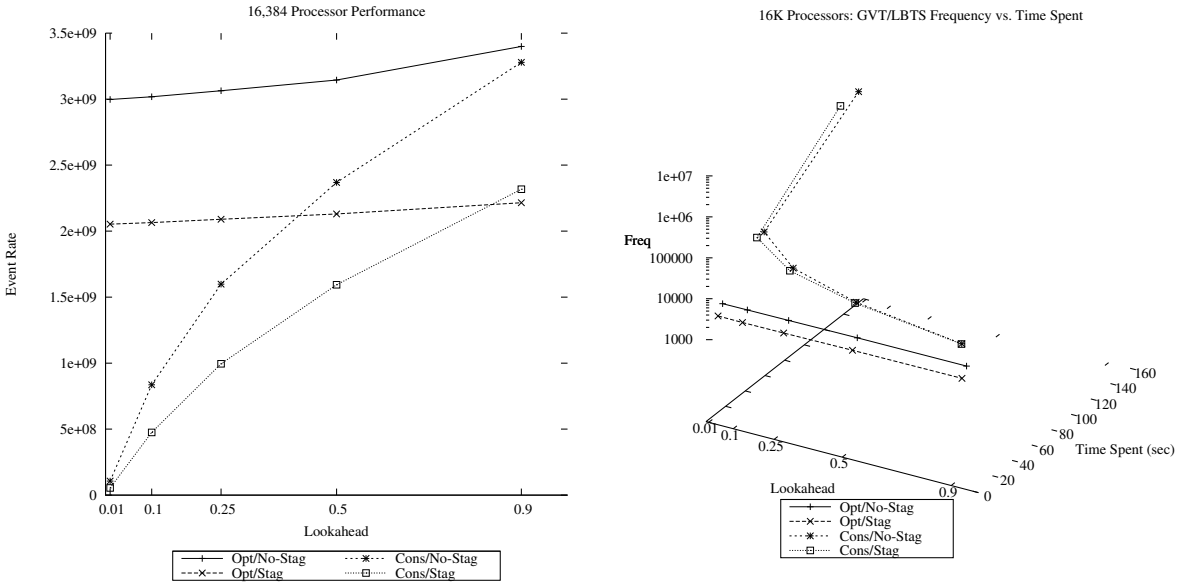
3.2 Observed Performance - 8K Cores



(a) Event rates (b) GVT/LBTS frequency and timing

Figure 3: 8,192 processor performance as a function of lookahead, and staggered vs. non-staggered variants of PHOLD models, with conservative and optimistic synchronization.

Figure 3(a) shows the observed event rates on 8,192 cores. Unlike the case with 1K cores, the results show clear effects of staggered vs. non-staggered workloads and optimistic vs. conservative execution across all lookahead values (except for the degenerate case of conservative execution with low lookahead). First, the improvement in conservative execution performance from low lookahead to small lookahead is not as dramatic. This is due to



(a) Event rates (b) GVT/LBTS frequency and timing

Figure 4: 16,384 processor performance as a function of lookahead, and staggered vs. non-staggered variants of PHOLD models, with conservative and optimistic synchronization.

the fact that the workload is divided across eight-fold as many processors as in the 1K case, thereby reducing the concurrency even with increased lookahead. The effect of reduced concurrency with conservative execution continues even to medium lookahead (0.5), and it finally attains the full potential performance reaching that of optimistic execution only on the largest lookahead (0.9). Another difference is that the difference in performance between conservative and optimistic execution becomes insignificant even with the largest lookahead.

The effects of staggered vs. non-staggered event workloads (that were negligible in the 1K case), however, begin to appear with 8K cores. Staggered case, as expected, constrains the overall concurrency offered by the model, thereby translating to reduced performance, compared to non-staggered (normal) event timestamp distribution. Optimistic execution incurs larger inefficiency from rollbacks, as evident in the 8K Proc/Stag case in Figure 5, showing a drop to 85% efficiency. Conservative execution, analogously, suffers increased GVT cost, as Figure 3(b) shows increasing time spent *per GVT computation*. Although the total time spent in GVT computation is the same across lookahead values of 0.5 and 0.9, the frequency is lower for lookahead of 0.9, indicating that each GVT computation more expensive with the larger lookahead.

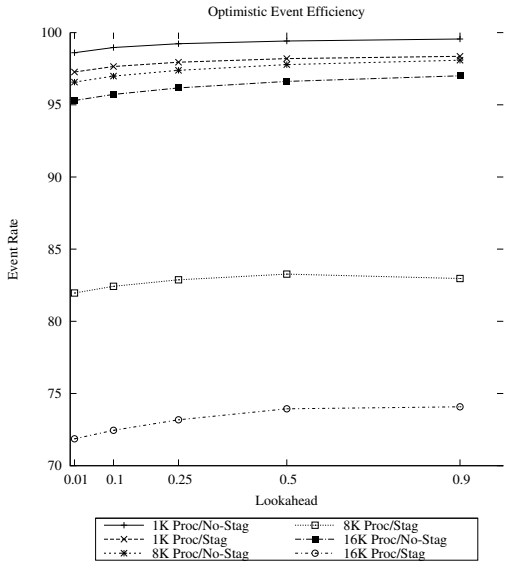


Figure 5: Optimistic event efficiency as a function of lookahead, and staggered vs. non-staggered variants of PHOLD models.

Overall, optimistic execution sustains high event rates across all lookahead values, showing the well-known resilience to lookahead values. Also observed is a slight increase in performance with larger lookahead values with normal (non-staggered) workload, since optimistic execution also is capable of benefiting from timestamps being farther into the future due to larger lookahead values. Conservative execution improves in performance (nearly linearly), starting from almost zero with low lookahead, all the way to exceeding the performance of optimistic execution (by up to 15% on staggered case) with high lookahead.

3.3 Observed Performance - 16K Cores

Figure 4(a) shows the observed event rates on 16,384 cores. The performance trends are almost similar to those in the case of 8K cores, with the only major difference being that conservative and optimistic executions are roughly identical with the largest lookahead, while optimistic execution performs better than conservative with all lower lookahead values.

The highest aggregate event rate observed in all the experiments in over 3 billion events executed per wall-clock second, with optimistic execution on 16K cores with a lookahead of 0.9 on the non-staggered workload. Event efficiency, however, is observed to be lower on the staggered workload on 16K cores, but not lower than 70% for any scenario. The GVT statistics also reflect relatively well-behaved behavior. Figure 4(b) shows the GVT computation statistics for the 16K cores case. For optimistic execution, the trend of increased blocked-time per GVT computation with increased lookahead values is observed. For conservative execution, the dramatic reduction in GVT computation time is also observed when lookahead is slightly improved (from 0.01 to 0.1), with less dramatic, but nearly linear, reduction in the number of LBTS computations. This implies that with higher lookahead and associated less frequent LBTS computations, that the model's execution become de-synchronized in simulated time across processors. Thus, there are some processors who are late to the global reduction operations which results in the increase time per LBTS. This is very to the OS Jitter problem encountered on supercomputer systems (Beckman, Iskra, Yoshii, Coghlan, and Nataraj 2008). A similar phenomenon is observed for optimistic.

Lookahead	1K cores	8K cores	16K cores
Low	Opt $>>>$ Cons	Opt $>>>$ Cons	Opt $>>>$ Cons
Small	Opt $\geq$ Cons	Opt $>$ Cons	Opt $>>$ Cons
Medium	Opt $<$ Cons	Opt $\geq$ Cons	Opt $\geq$ Cons
Large	Opt $<<$ Cons	Opt $\leq$ Cons	Opt $=$ Cons

Table 1: Qualitative summary of relative performance between conservative and optimistic execution

	1K cores		8K cores		16K cores	
Lookahead	Conservative	Optimistic	Conservative	Optimistic	Conservative	Optimistic
Low	S $=$ NS	S $=$ NS	S $=$ NS	S $<$ NS	S $=$ NS	S $<<$ NS
Small	S $=$ NS	S $=$ NS	S $<$ NS	S $<$ NS	S $<$ NS	S $<<$ NS
Medium	S $=$ NS	S $=$ NS	S $<$ NS	S $<$ NS	S $<<$ NS	S $<<$ NS
Large	S $=$ NS	S $=$ NS	S $<$ NS	S $<$ NS	S $<<$ NS	S $<<$ NS

Table 2: Qualitative comparison of relative performance on staggered (S) *vs*. non-staggered (NS) benchmarks

4 OVERALL QUALITATIVE SUMMARY

The quantitative results provided in the earlier section may be summarized in a qualitative manner as follows. Ignoring the precise quantiative levels of performance variation, and, instead, focusing on relative performance, Table 1 shows the performance of conservative execution relative to optimistic execution for the same benchmarks.

In the table, X$>>>$Y indicates X performs significantly better than Y, X$>>$Y indicates X performs noticeably better than Y, X$\geq$Y indicates X performs slightly better than or comparably to Y, X$<$Y indicates X performs slightly worse than Y, X$=$Y implies X's performance is roughly the same as that of Y within the margin of experiment error, and, similarly, X$<<$Y indicates that X performs poorly compared to Y.

In Table 2, the differences in the performance between the staggered and non-staggered cases are shown, for conservative and optimistic execution separately. The relative comparison operators for this table are similar to the ones used for Table 1 described earlier.

It is also important to note that the synchronization (GVT) frequency for optimistic execution can be relaxed up to the level allowed by available memory. Even with Blue Gene's low memory size, the memory seems to be sufficient to offset the larger blocked time (during GVT computation) relative to conservative synchronization.

5 CONCLUSIONS

We have attempted to provide guidance here to the greater PDES community on the conservative vs. optimistic execution performance on extant supercomputing systems. As our performance data suggest, the question of deciding on which parallel synchronization strategy to employ for discrete-event models depends on a number of factors and whose answer may not be immediately obvious. In particular, the model's inherent lookahead, how events are scheduled, event destination, and supercomputer hardware (network and processor performance) all come into play to determine performance.

Although clear winning arguments are difficult to make in general, a few inferences such as the following may be made based on the performance results reported here from the synthetic benchmarks:

- Low lookahead can make conservative execution perform very poorly (as one expects), with performance degrading significantly with increasing numbers of processors.
- Even with low lookahead, however, conservative execution may be acceptable for certain applications in which optimistic execution is either infeasible or prohibitively expensive to develop. From the experiments, optimistic execution is observed to be only 30$\times$ faster than conservative execution on 16,384 processors, in the worst case.

- Investment of effort in identifying/increasing lookahead in the model can provide significantly higher payback in terms of runtime performance, especially on a few thousand processors. From the experiments, the largest gain in performance for a small increase in lookahead was observed on 1,024 and 8,192 processors.
- The amount of additional gain from investments in identifying ever greater values of lookahead may not be sufficiently justified by the smaller increases in performance that would be obtained in return, unless the very highest performance levels are desired, regardless of the model development cost.
- For fine-grained models, it appears that optimistic execution is a clear winner, almost independent of the amount of lookahead in the application. This appears to hold regardless of the timestamp distribution across processors, as long as there is sufficient concurrency inherent in the application/model itself. Thus, if efficient rollback support is possible to develop for an application (e.g., using either incremental state saving, or using reverse computation), optimistic execution is a good choice for obtaining good runtime performance.

For the future, we plan to extend our performance study to include larger processor counts and different supercomputer systems, such as the Cray XT5, which has a significantly faster processor than the Blue Gene/L but without a dedicated global reduction network. This architectural difference may be expected to affect the phase-performance points for both conservative and optimistic protocols.

ACKNOWLEDGEMENTS

Blue Gene/L computational resources were provided by the Center for Computational Nanotechnology Innovations (CCNI) at Rensselaer Polytechnic Institute. Additionally, this paper has been partly supported by research sponsored by the Laboratory Directed Research and Development Program of the Oak Ridge National Laboratory, and co-authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the U.S. Department of Energy. Accordingly, the United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes.

REFERENCES

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

Bauer, D. W., G. Yaun, C. D. Carothers, M. Yuksel, and S. Kalyanaraman. 2005. Seven-o'clock: A new distributed gvt algorithm using network atomic operations. In *Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*, 39–48. Washington, DC, USA: IEEE Computer Society.

Bauer Jr., D. W., C. D. Carothers, and A. Holder. 2009. Scalable time warp on blue gene supercomputers. In *Proceedings of the 2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation*, 35–44. Washington, DC, USA: IEEE Computer Society.

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

Bellenot, S. 1992. State skipping performance with the time warp operating system. In *6th Workshop on Parallel and Distributed Simulation (PADS92)*, 24, 53–61.

Bryant, R. E. 1977, November. *Simulation of packet communication architecture computer systems*. Ph. D. thesis, MIT.

Carothers, C. D., K. S. Perumalla, and R. M. Fujimoto. 1999. Efficient optimistic parallel simulations using reverse computation. *ACM Transactions on Modeling and Computer Simulation* 9 (3): 224–253.

Chandy, K. M., and J. Misra. 1979, September. Distributed simulation: A case study in design and verification of distributed programs. In *IEEE Transactions on Software Engineering*, Volume 24, 440–452.

Dickens, P. M., D. M. Nicol, P. F. Reynolds, Jr., and J. M. Duva. 1996. Analysis of bounded time warp and comparison with yawns. *ACM Trans. Model. Comput. Simul.* 6 (4): 297–320.

D'Souza, L. M., X. Fan, and P. A. Wilsey. 1994. pgvt: an algorithm for accurate gvt estimation. *SIGSIM Simul. Dig.* 24 (1): 102–109.

Fujimoto, R. M., and M. Hybinette. 1997. Computing global virtual time in shared-memory multiprocessors. *ACM Trans. Model. Comput. Simul.* 7 (4): 425–446.

Gomes, F. 1996. *Optimizing incremental state-saving and restoration*. Ph. D. thesis, University of Calgary.

Gomes, Z. X. F., B. Unger, and J. Cleary. 1995. A fast asynchronous gvt algorithm for shared memory multiprocessor architectures. *SIGSIM Simul. Dig.* 25 (1): 203–208.

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

Jefferson, D. R. 1985. Virtual time. *ACM Trans. Program. Lang. Syst.* 7 (3): 404–425.

Lipton, R. J., and D. W. Mizell. 1999. Time warp vs. chandy-misra: A worst-case comparison. In *In Proceedings of the Sixth Workshop on Parallel and Distributed Simulation (Jan.)*, 43–52: Society for Computer Simulation.

Lubachevsky, B. D., A. Shwartz, and A. Weiss. 1991. An analysis of rollback-based simulation. *ACM Transactions on Modeling and Computer Simulation* 1 (2): 154–193.

Mattern, F. 1993. Efficient algorithms for distributed snapshots and global virtual time approximation. *Journal of Parallel and Distributed Computing* 18 (4): 423–434.

Nicol, D. M., and J. Liu. 2002. Composite synchronization in parallel discrete-event simulation. *IEEE Transactions on Parallel and Distributed Systems* 13:433–446.

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

Perumalla, K., and R. Fujimoto. 2001. Virtual time synchronization over unreliable network transport. In *PADS '01: Proceedings of the fifteenth workshop on Parallel and distributed simulation*, 129–136. Washington, DC, USA: IEEE Computer Society.

Perumalla, K. S. 2007. Scaling time warp-based discrete event execution to 104 processors on a blue gene supercomputer. In *CF '07: Proceedings of the 4th international conference on Computing frontiers*, 69–76. New York, NY, USA: ACM.

Perumalla, K. S. 2010a. $\mu\pi$: A scalable and transparent system for simulation mpi programs. In *In Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*.

Perumalla, K. S. 2010b. Reversible parallel discrete-event execution of large-scale epidemic outbreak models. In *In Proceedings of the 24th Workshop on Principles of Advanced and Distributed Simulation*.

Steinman, J. S. 1993. Breathing time warp. In *PADS '93: Proceedings of the seventh workshop on Parallel and distributed simulation*, 109–118. New York, NY, USA: ACM.

Xiao, Z., B. Unger, R. Simmonds, and J. Cleary. 1999. Scheduling critical channels in conservative parallel discrete event simulation. In *Proceedings of the 13th Workshop on Parallel and Distributed Simulation*, 20–28.

AUTHOR BIOGRAPHIES

CHRISTOPHER D. CAROTHERS is a Professor at RPI. His research interests are massively parallel processing, parallel I/O and parallel discrete-event simulation approaches and methods. His email address is <chrisc@cs.rpi.edu>.

KALYAN S. PERUMALLA is a Senior R&D Manager in the Computational Sciences and Engineering Division at the Oak Ridge National Laboratory. His email address is <perumallaks@ornl.gov>.