Appears in Proceedings of the ACM Computing Frontiers, Ischia, Italy, May 2007

Scaling Time Warp-based Discrete Event Execution to 10^4 Processors on a *Blue Gene* Supercomputer

Kalyan S. Perumalla
Oak Ridge National Laboratory
1 Bethel Valley Rd
Oak Ridge, Tennessee, USA
perumallaks@ornl.gov

ABSTRACT

Lately, important large-scale simulation applications, such as emergency/event planning and response, are emerging that are based on discrete event models. The applications are characterized by their scale (several millions of simulated entities), their fine-grained nature of computation (microseconds per event), and their highly dynamic inter-entity event interactions. The desired scale and speed together call for highly scalable parallel discrete event simulation (PDES) engines. However, few such parallel engines have been designed or tested on platforms with thousands of processors. Here an overview is given of a unique PDES engine that has been designed to support Time Warp-style optimistic parallel execution as well as a more generalized mixed, optimistic-conservative synchronization. The engine is designed to run on massively parallel architectures with minimal overheads. A performance study of the engine is presented, including the first results to date of PDES benchmarks demonstrating scalability to as many as 16,384 processors, on an IBM *Blue Gene* supercomputer. The results show, for the first time, the promise of effectively sustaining very large scale discrete event execution on up to 10^4 processors.

Categories and Subject Descriptors

I.6.1 [**Computing Methodologies**]: Simulation and Modeling, General – *parallel simulation*, *Time Warp*, *discrete event* C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures (Multiprocessors)

General Terms

Performance, Experimentation, Algorithms

Keywords

Parallel discrete event simulation, Time Warp, reverse computation, state saving, mixed-mode simulation

1. INTRODUCTION

Parallel discrete event simulation (PDES) finds use in many important application areas. While many PDES techniques have been studied for the past two to three decades, PDES execution on large processor counts remains to be demonstrated, especially in the context of fine-grained event execution. The fine-grained nature of event execution imposes tight constraints on PDES techniques with respect to scalability. Additionally, due to dynamic inter-processor communication patterns and runtime dependencies that are characteristic of PDES applications, the possibility of large-scale execution is to be explored by experimentation with actual software implementation and benchmarking of PDES engines. This paper fills this need, by evaluating and documenting the possible performance that can be achieved using a state-of-the-art PDES engine that incorporates a large set of PDES techniques in a single software implementation.

Specifically, the overall performance achieved by the PDES engine, µsik, is reported on up to 16,384 processors of an IBM Blue Gene supercomputer, with three important PDES synchronization mechanisms. Some of the recent PDES advancements exercised in this implementation include a reverse computation approach for realizing rollback, a fast global time synchronization algorithm that accounts for transient messages without using globally blocking barriers, a fast fossil collection method for (positive- and anti-) event reclamation, and efficient event-scheduling methods in mixed-mode (conservative plus optimistic) execution.

The rest of the paper is organized as follows. Motivation for large-scale, fine-grained PDES execution is presented in Section 2. A brief overview of PDES synchronization modes and relation to previous work is presented in Section 3. The experiment setup of the engine and synthetic benchmarks on the Blue Gene supercomputing platform are described in Section 4. Runtime performance results are presented in Section 5, followed by a summary and identification of additional future work in Section 6.

2. MOTIVATION

An increasing number of critical applications are evolving to warrant high end computing capabilities. Emergency planning & response, global/local social phenomena prediction & analysis, defense systems operations & effects, and sensor network-enabled protection/awareness systems are all representative application areas. Simulation-based applications in these areas include detailed vehicular and behavioral simulations, high-fidelity simulation of communication effects in large sensor networks, and complex models in social behavioral/agent-based simulations, to name just a few. The common core of these applications is their use of discrete event-based modeling approaches. Discrete event simulations involve evolving the states of the underlying entities (e.g., vehicles) in asynchronous fashion, in contrast to time-stepped simulations in scientific computing in which the entire system state is (logically) updated over fixed intervals (time-steps) of simulation time.

In their "next generation" levels of operation, these applications include scenarios characterized by their scale (several millions of simulated entities), their need for fast simulation of multiple scenarios (thousands of alternatives explored as fast as possible for reasoning, understanding and refining the models or solutions), and their (re)configuration based on large-sized dynamic data. When the application scenarios are scaled to large configurations of interest, they warrant the use of *parallel* discrete event simulation (PDES). For example, regional- or national-scale micro-simulation of vehicular traffic (e.g., for accurate computation of evacuation time or computation of energy consumption) involve the simulation of 10^6-10^7 road intersections and 10^7-10^8 vehicles. Many potential scenarios (e.g., evacuation routes, or alternative energy incentives) are to be evaluated. All such considerations together motivate the need for highly scalable PDES execution capabilities on high performance platforms.

However, computational scalability of PDES systems and techniques has never been tested on more than 10^3 processors. Execution on 10^3 processors is not yet mainstream in PDES application areas. Scalability beyond 10^3 processors has not been previously explored, and hence practicability and applicability of ultra-scale parallel execution of PDES applications has been unknown. Achievement and feasibility demonstration of PDES on very large configurations helps the simulation applications to begin to consider the use of such large scenarios in their respective application domains. The results of our efforts are motivated by such scenarios, and are aimed at helping expose the potential of the large-scale simulation capabilities for PDES applications.

3. BACKGROUND

A principal factor underlying the scalability challenge for PDES is that synchronizing event execution across processors is non-trivial, especially in keeping overheads low while achieving "global time stamp-ordered" execution. In applications of interest, such as microscopic vehicular simulation or packet-level Internet simulations, fine-grained event processing adds significantly to the challenge. In such simulations, event computation can consume very little wall-clock time, soon after which synchronization of event timestamps is required among the rest of the events in the parallel system.

For example, on current-day processors, packet-level models for Internet Protocol "packet handling" models take as little as 5-10μs per event for Internet simulations, and similarly, 10-50μs for updating vehicular states in microscopic models of vehicular transportation simulations. Correctness of results requires that events are executed to preserve *global* timestamp order among all events, which implies that global parallel synchronization must be very fast and efficient.

The rationale behind timestamp-ordered processing is that it permits the models to be accurately simulated, such that events are processed in the same order as their corresponding actions in the physical system. To enable such processing order, a simple local rule to follow is that a processor whose simulation time is at T should not receive events with timestamps less than T. Hence, advances of a processor's current time have to be coordinated and controlled carefully to prevent events appearing in processor's "past." This requirement gives rise to different synchronization approaches, and consequently, different algorithms.

3.1. Parallel Event Synchronization

In PDES, broadly three approaches are commonly used: conservative, optimistic, and mixed-mode synchronization. In all approaches, it is assumed that each individual entity in the simulation is represented as a *logical process* (LP), and logical processes communicate and synchronize with each other by sending/receiving time-stamped events.

Conservative: This approach always ensures safe timestamp-ordered processing of simulation events within each LP [1, 2]. In other words, an LP does not execute an event until it can guarantee that no event with a smaller timestamp will later be received by that LP. The guarantee of correctness is achieved by blocking the LP execution until all events destined to are generated and delivered from other LPs and it is safe to execute the LP's events.

Optimistic: This approach avoids "blocked waiting" by optimistically processing the events. When some events are later detected to have been processed in incorrect order (because new events with lower timestamps arrive from other LPs), the system invokes compensation code such as state restoration or reverse computation. The most well-known optimistic PDES algorithm is Time Warp[3].

Mixed-mode: This approach combines elements of the previous two. For example, sometimes it might help to have some parts of the application execute optimistically ahead, while other parts execute conservatively (e.g., see [4-6]). In such cases, a combination of synchronization techniques can be appropriate.

While several algorithms have been proposed in the literature for PDES, only a few have been shown to be inherently scalable, and even fewer have been actually tested on very large parallel platforms. Figure 1 shows the progress of PDES engine scalability over the past few decades. Examples of the largest discrete event simulation runs to date include (a) simulations of Internet-like networks using 1536 processors[7] (b) simulations of the PHOLD parallel simulation benchmark on 1024 processors[8, 9], and (c) simulations of discrete event models of particle-in-cell models on 512 processors[10]. Time Warp has previously been executed on increasing number of processors: 10^1 in 1980's[11], 10^2 in the 1990's, 10^3 in early 2000's. In this paper, we report scalability to 10^4 processors, which is represented by the data point listed for 2006.

Note that all PDES simulations are single parallel runs, in contrast to replicated independent runs employed by other parallel simulation approaches such as Monte Carlo methods. PDES is far being from embarrassingly parallel, with each simulation run utilizing all available processor count in a single session, with non-trivial synchronization operations among all processors.

For an overview of traditional as well as discussion of recent parallel/distributed simulation techniques and advances, the reader is referred to [12].

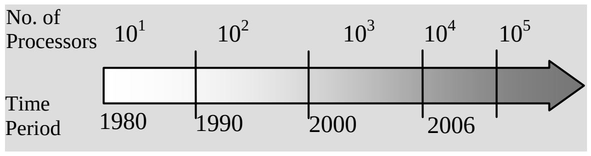

Figure 1: Progress of PDES engine scalability over time

In this context, research at our institution is focused along two tightly coupled directions. The first is in the development of a common, scalable core among PDES applications, and the second is in building applications on top of the scalable common core. In the first item, we have developed a range of novel PDES techniques (e.g., reverse computation-based optimistic synchronization, and scalable algorithms for virtual time synchronization) and also have developed software implementations of some of these. Earlier, these have been evaluated on up to 1536 processors, but their scalability to the scale of the systems such as the IBM Blue Gene with 10^4 processors has been unknown. In the second item, we have developed new models for some of the applications, designed with the overriding goal of parallel execution, right from inception. This paper is focused on the former, namely, on the performance of scalable PDES engines.

3.2. Related Work

The theory underlying the PDES synchronization protocols has been worked out well in the past. For example, the famous paper by David Jefferson on Time Warp[3] is seminal in many ways with respect to optimistic parallel simulation. The translation of the elegant theory into a practical implementation requires systems engineering work, such as efficient scheduling mechanisms, reducing memory footprints for optimal cache performance, and many additional critical details. The TWOS[13] and GTW[14] optimistic simulation systems are representative of early systems to address such details in order to take the concept of Time Warp to actual implementation. Specialized algorithms suited for symmetric shared memory multiprocessors helped increase the potential of Time Warp-style synchronization protocols up to the order of 10^2 processors. To move beyond 10^2 processors required additional systems work, as there remain many places where PDES engines needed to be optimized. Inefficiency, even in a few of them, could bring down the overall PDES execution efficiency. These include the costs of rollback support (state saving), global virtual time (GVT) computation, and fossil collection. Unknowns include questions such as how much overhead is involved in these operations when scaled to large number of processors, and doubts on whether they make the overall parallel execution worthwhile.

Among the few distributed memory Time Warp implementations reported are a small number of systems reported in mid 1990's on modest-sized cluster configurations such as up to 10^2 processors (e.g. Carothers'96 Ph.D. thesis), and the DSIM simulator tested recently on 10^3 processors[9].

In addition to the systems costs, there is the lingering concern about Time Warp instabilities: At what processor counts will we see the theoretically contemplated (e.g., [15]) instabilities? Will instabilities appear in scenarios at scales of interest in some of our important applications (e.g., a million logical processes for simulating one million road intersections)?

A large variety of PDES techniques were proposed over the past two decades or more (e.g., scheduling, flow control, bounded optimism, reverse computation, mixed mode, etc.), each of which is a complex endeavor in and by itself. Their use *together* in a single encompassing system such as μsik has not been evaluated. It remained unclear if at all such a unifying engine could in fact be developed without sacrificing performance.

4. PDES EXECUTION ON THE IBM *BLUE GENE*

Here we describe the details on the μsik engine for high performance PDES execution, and describe the porting process of the engine to the Blue Gene platform.

4.1. μsik PDES Engine

μsik is a PDES engine[8] that we designed with a unique "micro-kernel" approach to accommodating a wide range of PDES synchronization techniques, including conservative, optimistic and mixed-mode. Several applications have been built using PDES as their core, including: the SCATTER system for discrete event based modeling of large-scale vehicular mobility systems[16, 17]; the $PDES^2$ (PDES of PDES) system for virtual execution of large-scale PDES applications for performance prediction; physical system simulations such as discrete event modeling of Spacecraft Charging and other plasma physics phenomena[10, 18, 19]; and, Neurological simulations based on Hodgkin-Huxley models of neuron activity[20]. Currently, it is perhaps the only scalable engine capable of supporting mixed mode PDES execution.

μsik includes a unified system architecture for incorporating multiple types of simulation processes. The processes hold potential to employ a variety of synchronization mechanisms, and could even alter their choice of mechanism dynamically. Supported mechanisms include traditional lookahead-based conservative and state saving-based optimistic execution approaches. Also supported are newer mechanisms such as reverse computation-based optimistic execution and aggregation-based event processing, all within a single parsimonious application programming interface.

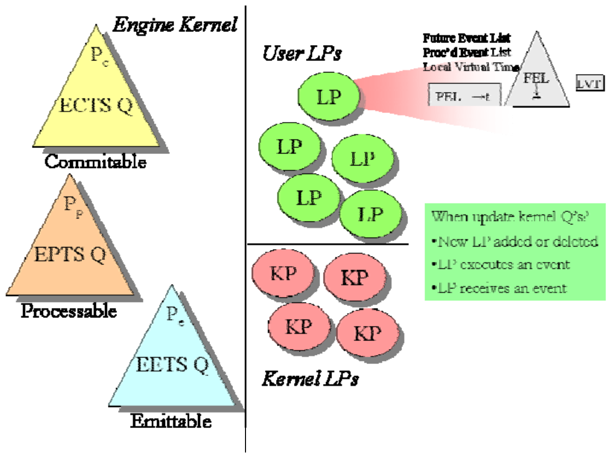

Figure 2: μsik engine data structures on each processor

The main internal data structures of µsik maintained on each processor are shown in Figure 2. The engine kernel maintains three main priority queues each of which contain references to logical processes. The queues are ordered by timestamp values of the processes. The Committable priority queue holds the logical processes ordered by the earliest timestamp at which events in each process could be committed (for example, fossil collected). Similarly, the Processable priority queue holds the logical processes ordered by their earliest events that could be executed. The Emittable priority queue holds the logical processes ordered by the earliest timestamps that could be generated in the future by each logical process. The least emittable timestamp at a processor is taken into account in computing the global virtual time (GVT) algorithm. In fact, a generalized version of GVT, namely, lower bound on incoming timestamp (LBTS) is used in global time synchronization in µsik.

Logical processes are user-defined model-level processes used in the PDES application (e.g., intersections in a road network). Additionally, the engine uses internal logical processes which themselves are also ordained by timestamp-ordered processing of their events, except that these internal "kernel LPs" of the engine are used for parallel communication and synchronization operations. For example, one kernel process per processor is created to house anti-messages for Time Warp, and to fossil collect them efficiently at runtime when GVT sweeps past them (simply by "committing" the anti-messages). Flow control is also implemented in these kernel processes – message sends that fail due to destination buffers being full will be held and re-sent by the kernel process mapped to that destination processor.

4.2. The Blue Gene Supercomputer

The scaling experiments reported in this paper were performed on the Blue Gene supercomputer, called the Blue Gene Watson (BGW), housed at the IBM T. J. Watson Research Center. As with the Blue Gene architecture, it consists of 16 racks, each rack consisting of 1024 nodes, each node consisting of two cores, each core being of type PowerPC-440. In a compute node (CN) configuration, one core computes while the other core services the network. In a virtual node (VN) configuration, both cores are used as compute nodes, giving an effective total of 32,768 processors that can be used in the largest computing configuration on the BGW. As of this writing, BGW is rated second in the Top 500 supercomputer installations.

4.3. Porting to the Platform

µsik is designed to execute on top of a range of inter-processor networks such as TCP/IP, Myrinet/GM, System V shared memory, MPI, and *any combinations* of the same. To permit maximum flexibility, selection of network routes can be postponed all the way to runtime, as opposed to being fixed at compile time. This flexibility in our software created problems on porting to the Blue Gene, since there is no support for socket libraries, shared memory and so on. We addressed this problem by introducing conditional compilation macros to compile out any and all references to sockets and multi-threading libraries in the µsik code. µsik is written using ANSI C and C++. We used the Blue Gene's native compilers to compile µsik code: `blrts_xlc` (for C) and `blrts_xlC` (for C++). Some of the important compiler flags that we used are:

```
 -O2 –qtune=440 –qarch=440d .
```

The software implementation was also customized for the Blue Gene execution environment by setting certain compile-time macro parameters of µsik (such as MAXPE for maximum number of processors) as appropriate, and eliminating or circumventing all $O(n^2)$ data structures, where n is the number of processors used in simulation.

4.4. Debugging Execution

We experimented with the use of the `-O4` optimization flag with the goal of reaping its performance improvements. However, its use resulted in incorrect execution in which the processors failed to synchronize, so we had to revert back to the `-O2` optimization flag. Unfortunately, we did not have time on the Blue Gene to spend on debugging exactly where `-O4` was creating problems in the code. In future work, we intend to debug and resolve this problem to take advantage of `-O4` optimizations (such as compiler assistance to exploit dual floating point units that are especially available on the Blue Gene).

4.5. Debugging Performance

We were bitten by the well known performance pitfall on the Blue Gene, namely, that a double word misalignment trap induces wasted CPU time with thousands of cycles of delay in servicing the misaligned loads/stores. We ran into this problem early on when porting µsik to the BGW. Processing time per event dramatically increased, from 8µs to 67µs per event, and overall event processing rate plummeted! We fixed this by padding structures in time management header fields to be integer multiples of 8 bytes. Before the fix was applied, coerced casts of structures for layered event header processing made the execution trap on double precision timestamp fields. Locating the exact source of the problem in source code was extremely challenging as the offending memory addresses were difficult to trace back to source code, even when symbolic debugging information was turned on during compilation. We were able to track down the source of the problem via the web-based Remote Administration Services (RAS) interface of the Blue Gene, in combination with object dumps.

4.6. Overall Parallel Execution Challenge

Efficient implementation of optimistic parallel simulation engine (e.g., using Time Warp) is already a complex endeavor. The complexity increases when the functionality of mixed-mode execution is added, and increases even further when newer techniques such as reverse computation are incorporated for low-overhead rollback. Software engineering complexity is amplified because execution on very large scale induces wider code coverage, exposure of memory leaks and other inefficiencies. Our porting of the µsik engine to Blue Gene scale indeed touched upon our solutions to all such challenges in our implementation. Over the course of the experiments, our algorithms were found to be well suited to address these challenges. For example, our scalable time synchronization algorithm[21] for computation of Global Virtual Time (GVT) or Lower Bound on Incoming Time Stamp (LBTS) served well to continue scaling from 10^1 to 10^4 processors seamlessly, taking into account transient messages without using blocking barriers. This algorithm works by executing iterations of reductions across all processors, each reduction attempting to account for any time-stamped messages in flight unaccounted for by a distributed snapshot. When after any iteration it is found that there are no more transient messages that remain unaccounted for in flight, the globally reduced timestamp value across all processors gives the global virtual time. Although the theoretical logarithmic complexity of each iteration in the algorithm is known, the number of iterations is not necessarily bounded, which depends on runtime behavior (e.g., network speed). Nevertheless, it was empirically confirmed from the results of the experiments that the number of iterations was always small (averaging less than 2). This is probably attributable to the fact that the Blue Gene architecture has a set of fast interconnection networks for global communication.

Many other optimizations were brought into play by the execution on 10^4 processors (e.g., scalable fossil collection of anti-message handles via special logical processes). The optimistic runs, for example, used reverse computation as the rollback mechanism, which eliminated the significant memory overheads of state saving needed for saving logical process state. We believe the reduced footprint due to reverse computation helped enable the low per-event overhead observed in our experiments. Further confirmation work is needed (e.g., by instrumentation for cache miss statistics), but our prior experience along these lines[22] indicates that this is probably the case. Similarly, the memory footprint reduction is also aided by fast, logarithmic operation on fossil collection data structures for usual (positive) event processing as well as for anti-message handling. Since a major source of memory overheads is event buffers, and our kernel-LP implementation for flow control and anti-messages help reclaim anti-messages as fast as possible, both at the sender side as well as at the receiver side across distributed memory boundaries.

5. PERFORMANCE RESULTS

Here we present the runtime performance results scaling to 16384 processors by using up to 8 racks of the BGW in virtual node (VN) mode (8 racks × 1024 nodes per rack × 2 processors per node).

5.1. Benchmark Application

For the performance study, we used the PHOLD application, which is a de facto standard PDES benchmark. The PHOLD application helps test functionality and performance simultaneously, while providing easily controllable configurations that are easy to understand yet challenging to parallelize effectively. It provides for a powerful way to control a wide variety of PDES application characteristics, and often serves as a worst case benchmark to help debug, verify and stress-test PDES engines. This benchmark is widely used in PDES as an easily developed, yet complex enough, test application.

In PHOLD, logical processes (LPs) juggle a fixed number of time-stamped events among themselves; each LP sends a time-stamped event to a randomly selected destination LP with an exponentially distributed simulation time increment. For the experiments, we chose a configuration that included 1 million LPs and 10 million events being juggled by all LPs (which implies that there are at least 10 million events at any given moment in the entire simulation). Although µsik has earlier been used to simulate scenarios with 1 million LPs and 1 billion events[8] on the IBM DataStar machine at the San Diego Supercomputing Center, USA, we chose a more modest sized configuration for the IBM Blue Gene as a preliminary effort. The safety of lower number of events was chosen as a guaranteed fit within the low memory availability on each BGW node; this was done in order to minimize wastage of failed runs within the precious, limited allocation hours to which we had access on the BGW machine.

In order to allow for concurrency, every event is scheduled with a minimum simulation time increment of 1.0, which is added to the exponentially distributed time increment with a mean of 1.0. Destination logical process is chosen with 90% locality (i.e., 10% of events cross processor boundaries).

In conservative mode, events are executed under strictly safe timestamp order. Every LP processes all its events in non-decreasing order of event timestamps. Execution is blocked for safety if/as necessary, and safety and progress are both ensured using global virtual time computation. In optimistic mode, LPs process their events almost independently of progress of other LPs' timelines. Time Warp-based rollback schemes (using anti-messages and reverse computation) are used to correct any incorrect execution, such that eventually the entire execution guarantees total global timestamp order, giving the same results as a conservative parallel execution. In mixed mode, all LPs with even identifiers execute conservatively, and all LPs with odd identifiers execute optimistically.

5.2. Problem Scaling Method

In our tests, we used the strong scaling method when scaling to larger number of processors. The problem size remained fixed while the number of processors was increased. We have also used weak scaling in our experiments in the past, and envision experimenting with weak scaling in future on the Blue Gene as well. We believe that strong scaling represents a more challenging problem than weak scaling, and consequently, we expect even better scalability results with weak scaling, compared to the strong scaling results reported here.

As an illustrative point, it is well known that optimistic methods such as Time Warp exhibit instabilities resulting in poor performance on large numbers of processors when the event load per processors decreases. Strong scaling reduces event load as number of processors is increased to a point where the event load became too low per processor on 8 racks. Thus, as expected, we observed this phenomenon of low performance when scaling from 8192 to 16384 processors. We believe that performance of optimistic and mixed mode execution would improve on 16384 processors, and beyond, when the number of logical processes and events per logical process is scaled according to weak scaling. Time allocation constraints on the BGW machine prevented us from performing weak scaling and other experiments. We intend to perform these additional experiments in the near future.

The observed runtime speedup is shown in Figure 3. The chart shows that conservative mode scaled well all the way up to 16384 processors of 8 racks. Mixed mode and optimistic modes scaled fairly well up to 8192 processors of 4 racks. At 8 racks, optimistic execution became sufficiently lightly loaded to encourage foray into unstable optimistic regions, which resulted in overheads overtaking the gains from optimism. Mixed mode execution performed better than purely optimistic execution because of the constraints by conservative processes.

5.3. Runtime Performance

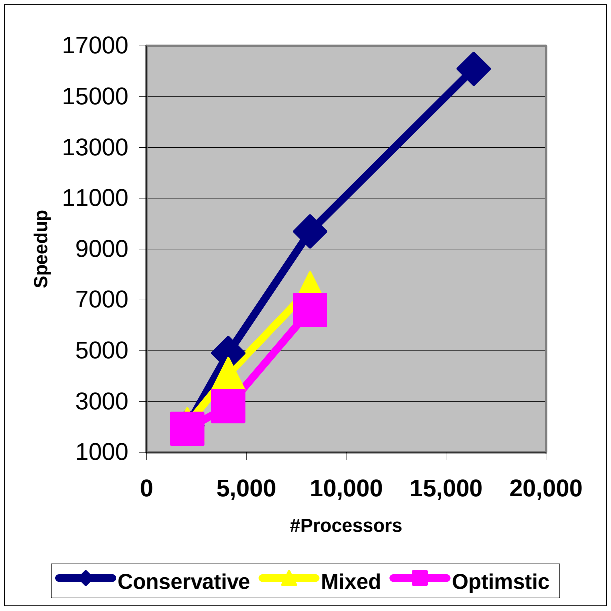

Figure 3: Parallel execution speedup with strong scaling

The flip side of degraded performance of Time Warp beyond 8192 processors is the surprising positive outcome that Time Warp in fact is feasible on up to 8192 processors with only a few hundred LPs (128 in this case) per processor. While other previous work has shown that Time Warp can work efficiently with very large number of LPs (e.g., one million LPs on 4 processors[23]), our results show that significant level of strong scaling is also now possible and conceivable to be exploited in applications that need it.

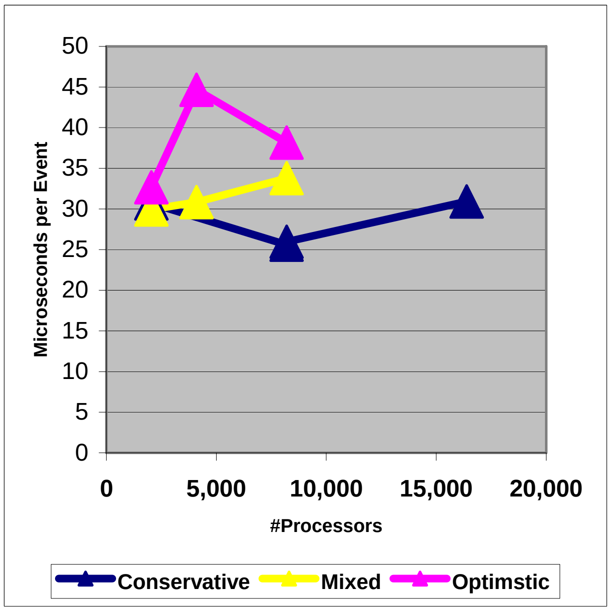

Figure 4: Average time to process each event

The average time to process each event is shown in Figure 4. This time includes application-specific computation as well as engine-induced overheads for synchronization, event scheduling, allocation/de-allocation and other operations. The results show very low event overhead, even for fine grained event computation, placing it within effective reach of fine-grained PDES applications.

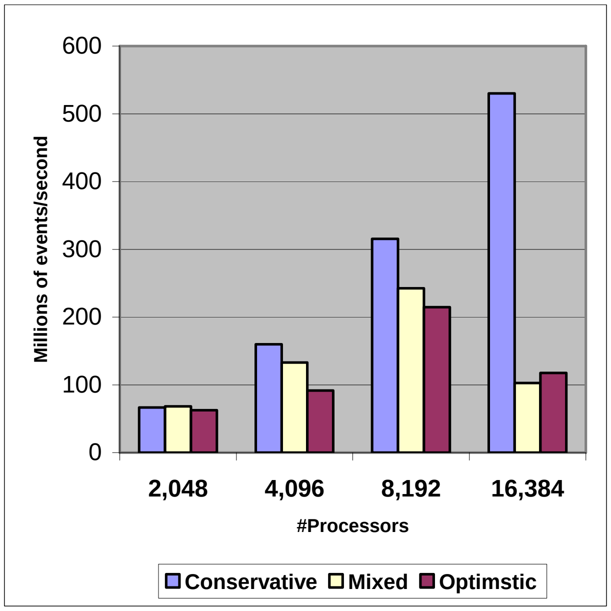

Figure 5: Aggregate rate of event execution across all processors combined

The total event processing rates achieved by all processors combined are shown in Figure 5. These represent some of the largest event rates ever registered for PDES. Conservative simulation delivers the largest aggregate event rate of 530 million events per wall clock second using 16384 processors. The largest optimistic simulations deliver 214 million events per wall clock second using 8192 processors. The largest mixed mode simulations deliver 243 million events per second using 8192 processors.

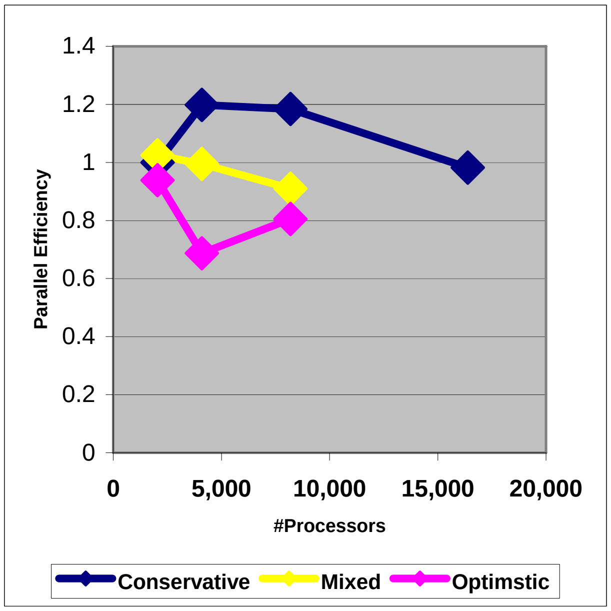

Figure 6: Efficiency of parallel execution, assuming full efficiency for 2000 processor execution

Figure 6 shows the parallel execution efficiency, normalized with the assumption of unit efficiency on 2000 processors. Super-linear speedup is observed by conservative parallel simulations. This is due to either normalization with 2000 processors (which itself could have an absolute efficiency of less than unity), and/or due to better caching and lower event scheduling overheads when the number of logical processes per processor decreases with increasing number or processors.

6. SUMMARY AND FUTURE WORK

While the previous largest conservative parallel simulation has been limited to 1536 processors, here we demonstrated that PDES with conservative execution can scale with excellent speedup on up to 16384 processors. Similarly, the capability for Time Warp-style of optimistic parallel simulation has been improved from the previous largest configuration of 1033 processors to a new level of 8192 processors. Optimistic simulation, however, exhibited degraded speedup beyond 8192 processors (on 16384 processors), and requires additional work to tune the system (e.g., for minimizing the number of LBTS computations) or experiment with weak scaling.

The experiments also demonstrated the largest mixed mode simulation to date. The previous largest mixed mode simulations that we are aware of are only limited to using a core that is only capable of mixed mode execution but not fully time synchronized. This is the execution of the JSAF federation using a High Level Architecture Run Time Infrastructure implementation on up to 1024 processors[24]. True time-synchronized mixed mode execution has been performed earlier, but on far fewer processors (up to 16 processors) [25-27].

The feasibility of efficient, low-overhead execution of PDES on large-scale parallel platforms opens the possibility for executing very large configurations of important PDES applications. It is clear that porting, tuning and testing the PDES engine is by itself a major task, and porting applications is much more challenging. As follow on to the work reported here, we intend to port our large-scale PDES applications being built over µsik and study their performance as well, by building on the promise provided by this engine benchmarking study.

ACKNOWLEDGEMENTS

This paper has been 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.

This effort has been supported by research sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, managed by UT-Battelle, LLC, for the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

The author thanks Dr. Sadaf Alam of ORNL for helping with information on compilation and execution procedures on the Argonne National Laboratory's Blue Gene machine. The author wishes to acknowledge the gracious assistance of IBM researchers Drs. Fred Mintzer, John Gunnels and Robert Walkup for facilitating access to the IBM Blue Gene Watson. The BGW Consortium Days program[28] by IBM provided us with an opportunity in which our PDES engines and applications could be ported to and tested on very large scale parallel environment.

REFERENCES

[1] K. M. Chandy and J. Misra, "Asynchronous Distributed Simulation via a Sequence of Parallel Computations," *Communications of the ACM,* vol. 24, pp. 198-205, 1981.

[2] K. M. Chandy and J. Misra, "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs," *IEEE Transactions on Software Engineering,* vol. SE-5, pp. 440-452, 1978.

[3] D. Jefferson, "Virtual Time," *ACM Transactions on Programming Languages and Systems,* vol. 7, pp. 404-425, 1985.

[4] H. Rajaei, R. Ayani, and L.-E. Thorelli, "The Local Time Warp Approach to Parallel Simulation," in *Workshop on Parallel and Distributed Simulation,* San Diego, California, United States, 1993.

[5] K. S. Perumalla, "µsik - A Micro-Kernel for Parallel/Distributed Simulation Systems," in *Workshop on Principles of Advanced and Distributed Simulation,* Monterey, CA, USA, 2005.

[6] V. Jha and R. Bagrodia, "A unified framework for conservative and optimistic distributed simulation," in *Workshop on Parallel and Distributed Simulation*, 1994, pp. 12-19.

[7] R. M. Fujimoto, K. S. Perumalla, A. Park, H. Wu, M. Ammar, and G. F. Riley, "Large-Scale Network Simulation -- How Big? How Fast?," in *Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 2003.

[8] K. S. Perumalla, "Scalable and Flexible Parallel/Distributed Simulation Systems: A Micro-Kernel Approach," Oak Ridge National Laboratory, Technical Memorandum 2005/12/01 2005.

[9] D. Chen and B. K. Szymanski, "DSIM: Scaling Time Warp to 1,033 Processors," in *Winter Simulation Conference*, Orlando, FL, 2005.

[10] K. S. Perumalla, R. M. Fujimoto, and H. Karimabadi, "Scalable Simulation of Electro-magnetic Hybrid Codes," in *6th International Conference on Computational Science*, Reading, UK, 2006, pp. 41-49.

[11] D. R. Jefferson, B. Beckman, F. Wieland, L. Blume, M. DiLorento, P. Hontalas, P. Reiher, K. Sturdevant, J. Tupman, J. Wedel, and H. Younger, "The Time Warp Operating Systems," in *11th Symposium on Operating Systems Principles*. vol. 21, 1987, pp. 77-93.

[12] K. S. Perumalla, "Parallel and Distributed Simulation: Traditional Techniques and Recent Advances," in *Winter Simulation Conference,* Monterey, California, USA, 2006.

[13] D. O. Rich and R. E. Michelsen, "An Assessment of the Modsim/TWOS Parallel Simulation Environment," in *Proceedings of the 1991 Winter Simulation Conference,* 1991, pp. 509-518.

[14] S. Das, R. M. Fujimoto, K. Panesar, D. Allison, and M. Hybinette, "GTW: A Time Warp System for Shared Memory

Multiprocessors," in *Proceedings of the 1994 Winter Simulation Conference*, 1994, pp. 1332-1339.

[15] B. D. Lubachevsky, A. Shwartz, and A. Weiss, "Rollback Sometimes Works... If Filtered," in *Proceedings of the 1989 Winter Simulation Conference*, 1989, pp. 630-639.

[16] K. S. Perumalla, "A Systems Approach to Scalable Transportation Network Modeling," in *Winter Simulation Conference*, Monterey, CA, 2006.

[17] K. S. Perumalla and B. Bhaduri, "On Accounting for the Interplay of Kinetic and Non-kinetic Aspects in Population Mobility Models," in *European Modeling and Simulation Symposium*, Spain, 2006.

[18] Y. Tang, K. S. Perumalla, R. M. Fujimoto, H. Karimabadi, J. Driscoll, and Y. Omelchenko, "Optimistic Parallel Discrete Event Simulations of Physical Systems using Reverse Computation," in *Workshop on Principles of Advanced and Distributed Simulation,* Monterey, CA, USA, 2005.

[19] Y. Tang, K. S. Perumalla, R. M. Fujimoto, H. Karimabadi, J. Driscoll, and Y. Omelchenko, "Optimistic Simulations of Physical Systems using Reverse Computation," *SIMULATION: Transactions of The Society for Modeling and Simulation International,* vol. 82, pp. 61-73, 2006/01/01 2006.

[20] C. J. Lobb, Z. Chao, R. M. Fujimoto, and S. Potter, "Parallel Event-Driven Neural Network Simulations Using the Hodgkin-Huxley Neuron Model," in *Workshop on Principles of Advanced and Distributed Simulation*, Monterey, CA, 2005.

[21] K. S. Perumalla and R. M. Fujimoto, "Virtual Time Synchronization over Unreliable Network Transport," in *Workshop on Parallel and Distributed Simulation*, 2001.

[22] C. Carothers, K. S. Perumalla, and R. M. Fujimoto, "Efficient Optimistic Parallel Simulations using Reverse Computation," *ACM Transactions on Modeling and Computer Simulation,* vol. 9, pp. 224-253, 1999/07/01 1999.

[23] Y. Garrett, D. C. Christopher, and K. Shivkumar, "Large-Scale TCP Models Using Optimistic Parallel Simulation," in *Proceedings of the seventeenth workshop on Parallel and distributed simulation*: IEEE Computer Society, 2003.

[24] D. M. Davis, R. F. Lucas, P. Amburn, and T. D. Gottschalk, "Joint Experimentation on Scalable Parallel Processors," *Journal of the International Test and Evaluation Association,* vol. Summer 2005, 2005/05/01 2005.

[25] V. Jha and R. Bagrodia, "A Unified Framework for Conservative and Optimistic Distributed Simulation," in *Proceedings of the 8th Workshop on Parallel and Distributed Simulation*, 1994, pp. 12-19.

[26] R. Bagrodia and W.-T. Liao, "Maisie: A Language for the Design of Efficient Discrete-Event Simulations," *IEEE Transactions on Software Engineering,* vol. 20, pp. 225-238, 1994.

[27] R. L. Bagrodia, "Iterative Design of Efficient Simulations using Maisie," in *Proceedings of the 1991 Winter Simulation Conference*, 1991.

[28] IBM, "Blue Gene Watson Consortium Days," F. Mintzer, Ed. Yorktown, NY, USA, 2006.