A CONCURRENT MULTI-TIER PRIORITY QUEUE FOR MULTITHREADED OPTIMISTIC PARALLEL DISCRETE EVENT SIMULATION

Matthew DePero
CSE Department, Miami University
Oxford, OH 45056, USA.
email: deperomm@miamiOH.edu

Dhananjai M. Rao
CSE Department, Miami University
Oxford, OH 45056, USA.
email: raodm@miamiOH.edu

KEYWORDS

PDES, Time Warp, Multithreading, Lockfree, Concurrent Pending Event Set, Threadsafe Datastructure

ABSTRACT

Multithreaded Parallel Discrete Event Simulation (PDES) conducted using emerging shared-memory many-core CPUs presents capacity for even greater performance. However, realizing good performance is challenging and is contingent on the data structures used for managing pending events. Accordingly, we propose a lightweight, thread-safe priority queue called `3tSkipMT` for managing pending events. Our design takes advantage of contemporary synchronization primitives, including atomics and lock-free instructions to ensure good performance. `3tSkipMT` has been incorporated into a significantly redesigned version of a parallel simulator called MUSE. Our investigations identify several critical design issues and we discuss novel solutions to address them. The effectiveness of the proposed solution has been assessed using the standard PHOLD benchmark. The experiments show that our approach outperforms other state-of-the-art methods by achieving good, near linear speedups of ~7.5× on 8 cores.

INTRODUCTION

Discrete event simulation is an important, scientific methodology that is used for analysis and design in many fields. The rapid advancement in hardware has catalyzed the widespread use of Parallel Discrete Event Simulation (PDES). Specifically, optimistic PDES have grown in importance as they provide improved parallelism and performance over conservative simulations in many scenarios (Jafer et al., 2013). Particularly, the rapid increase in the number of CPU-cores in machines, some with as many as 72 cores, has spurred research into enabling efficient, multithreaded optimistic PDES by leveraging shared-memory parallelism.
Shared-memory parallelism) offers two key advantages over conventional distributed-memory parallelism. First, it eliminates communication overheads associated with exchanging events between parallel processes. Figure 1 presents an example of an optimistic PDES run on one compute node using Cross Memory Attach (CMA) enabled OpenMPI library (Rao, 2018). CMA uses Linux-kernel capabilities to directly move data from one process to another using shared memory. Nevertheless, despite effective partitioning, with increasing interaction between processes, the communication overheads exceed 65% as shown in Figure 1. This conspicuous overhead can be significantly reduced through shared-memory PDES realized via multithreading (Rao, 2018). Second, multithreading reduces memory requirements by allowing a single copy of an event to be readily shared between multiple threads.

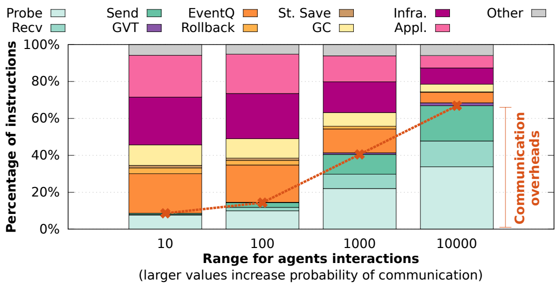


Figure 1: Communication overheads in PDES

Challenges with efficient multithreaded PDES

In practice, realizing the aforementioned advantages of multithreaded PDES is challenging due to a myriad of contention, consistency, and performance issues with multithreading. A crucial issue is efficiently managing the Pending Event Set (PES), *i.e.,* set of timestamped events to be processed chronologically. The PES is the central data structure that experiences rapid enqueue, dequeue, and cancellation operations concurrently from many threads. Avoiding race conditions leads to heavy contention, thereby rapidly diminishing the realized speedup. For example, even with a recent multithreaded optimistic PDES simulator from Ianni et al. (2018), the observed speedup was only ~3× on 8 cores, instead of the expected ~8×. Consequently, designing efficient multithreaded PES data structures for PDES is a challenge and is an active area of research (Rao and Higiro, 2019).

Overview of proposed research

In this study, we propose and explore a novel three-tier data structure called `3tSkipMT` to enable efficient multithreaded PDES. The key advantages of its design include: ❶ multi-tiered concurrent skip lists for reducing contention within the data structure and ❷ the use of lock-free operations to minimize the computational overheads due to contention. The `3tSkipMT` has been incorporated into a significantly redesigned version of a parallel simulator called MUSE, to enable multithreaded PDES on shared-memory platforms. Our investigations highlight several critical design issues in enabling efficient PDES. The effectiveness of the proposed solution has been assessed using the well-established PHOLD benchmark. Our solution achieves close to linear speedup of up to ~7.5× on 8 cores.

BACKGROUND

An optimistic Parallel Discrete Event Simulation (PDES) is organized as a set of interacting Logical Processes (LPs). LPs are partitioned to execute on multiple threads or processes, running on independent compute-elements such as: compute nodes, CPU-cores, or even GPU-cores. LPs interact with each other by exchanging *virtual time* stamped events. Unlike in conservative PDES, in an optimistic PDES, the LPs do not explicitly synchronize with each other. Instead, LPs process events as soon as they are received, change their internal states and schedule new events into the future asynchronously. This approach significantly improves concurrency but leads to causality errors in a PDES.

Time Warp is the most widely used synchronization protocol for optimistic PDES. In Time Warp, each LP maintains its Local Virtual Time (LVT), tracking the timestamp of the most recently processed event and saves its states keyed with the corresponding LVT. Causal violations are detected by an LP when it receives a *straggler* event, *i.e.,* an event with timestamp before LVT, indicating incorrect order of event processing. The LP then rolls back to an earlier saved state, sends out anti-messages to cancel events sent after the straggler timestamp, and reprocesses events in correct timestamp order. Anti-messages trigger rollback operations as needed in other LPs. Rollback recovery requires maintaining states and events which are periodically garbage collected based on Global Virtual Time (GVT) estimates. GVT tracks the lowest LVT to which an LP could rollback.

Pending Event Set (PES) for Time Warp

The Pending Event Set (PES) that contains events to be processed. It strongly influences the overall performance of a Time Warp based optimistic PDES because it is used to perform the following four key operations – ❶ **Enqueue**: This operation adds one or more events to the pending event set, ❷ **Peek**: Identify the LP with the lowest timestamped event to be processed, ❸ **Dequeue**: In contrast to peek, this operation removes the next events from the data structure for processing by an LP, and ❹ **Cancel**: This operation is used as part of rollback recovery process to remove all pending events sent by a given LP (LP_{sender}) to another LP (LP_{dest}) at-or-after a given virtual time ($LVT_{rollback}$).

RELATED RESEARCH

The Pending Event Set (PES) is the primary data structure for managing events to be processed in a PDES. Hence, there is a breadth of literature offering a variety of PES implementations, each with its own pros and cons. One of the original data structures used for this purpose was the Calendar Queue (CQ) by Brown (1988). It is structured as an array of buckets each associated with a fixed span of virtual time. The width, or time span, associated with a bucket must be carefully chosen, but when optimal, CQ allows for amortized constant time enqueue of events and constant time retrieval of the minimum timestamp event.
A popular variant of CQ is the Ladder Queue (`LadderQ`) proposed by Tang et al. (2005). It outperforms several popular structures, including the CQ. This is done through the reducing of bucket management overheads by allowing buckets to be split dynamically when a bucket's size exceeds a threshold. Bucket splitting may result in a new rung to be added to the latter queue. A key limitation of the `LadderQ` arises in optimistic PDES – a complete linear scan of all events is needed for event cancellations during rollbacks. The 2-Tier Ladder Queue (2tLadder) proposed by Higiro et al. (2017) addresses this issue by further subdividing a bucket. For multithreaded optimistic PDES, Gupta and Wilsey (2014) explore the use of lock-free queues for the bottom of the `LadderQ` used for managing pending events.
The 3-tier Heap (`3tHeap`) was recently proposed for PES (Rao and Higiro, 2019). This data structure conceptually has 3-tiers, with the top tier heap sorting agents based on their next available event, the second tier heap sorting events for that agent into individual timestamps, and the third tier vector holding the individual events at each timestamp. The `3tHeap` has shown to achieve good performance gains over `2tLadderQ`, in a broad range of settings, especially in optimistic PDES with a large number of *concurrent* events (*i.e.,* events with the same timestamp scheduled to the same LP).
Hence, in this research, we build on the multi-tiered philosophy of `3tHeap`. However, the `3tHeap` is designed or a single threaded operations and does not permit concurrent operations from multiple threads. In contrast, we propose a novel `3tSkipMT` data structure that enables efficient concurrent multithreaded operations for high performance optimistic PDES.
Wang et al. (2014) discuss issues of enabling effective, multithreaded PDES and show performance improvements on a variety of CPU architectures. Multithreaded optimistic PDES have been investigated and reported by a team of researchers in Marotta et al. (2017) and Ianni et al. (2018). Their simulator is called Ultimate Share Everything (USE), which is similar to our proposed methodology. The USE simulator utilizes a conflict-resilient, lock-free calendar queue. Our `3tSkipMT` is similar in that it uses a lock-free approach but in contrast, it uses a multi-tiered approach along with a concurrent skip list. Similar to this research, every one of the aforementioned investigations used the PHOLD performance benchmark for assessments.

MULTITHREADED PARALLEL SIMULATOR

In this study we have enabled multithreaded, Time Warp synchronized PDES by significantly redesigning an existing simulator called MUSE Rao (2018). We have adopted the approach of enhancing MUSE because of several reasons. First, MUSE already provides a well-designed Application Program Interface (API) for modeling. Adhering to its API enables reusing existing models, including the PHOLD benchmark thereby easing verification and validation. Second, MUSE's simulation-kernel already provides the infrastructure for distributed, consistent, and verifiable execution of optimistic PDES. Reusing this functionality minimizes development overheads. Lastly, but importantly, MUSE already includes `3tHeap` which is currently one of the fastest data structures for managing Pending Event Set (PES). Even though `3tHeap` is designed for single-threaded operation, it is still an excellent "point of reference" for performance comparisons.

Figure 2 illustrates an overview of a multithreaded MUSE PDES. The simulation consists of a given number of simulation-kernel threads. The number of threads is specified as a command-line argument. There is inherently no upper limit on the number of threads. However, for best performance, the number of threads should be no more than the number of CPU-cores. Each kernel thread is responsible for enabling a Logical Process (LP) to operate on its next set of events. The threads operate using a shared scheduler-queue called `3tSkipMT` that manages the PES. The `3tSkipMT` is centralized (*i.e.,* manages events for all LPs) and enables multiple threads to concurrently operate on independent LPs. The next section presents details on the `3tSkipMT`.

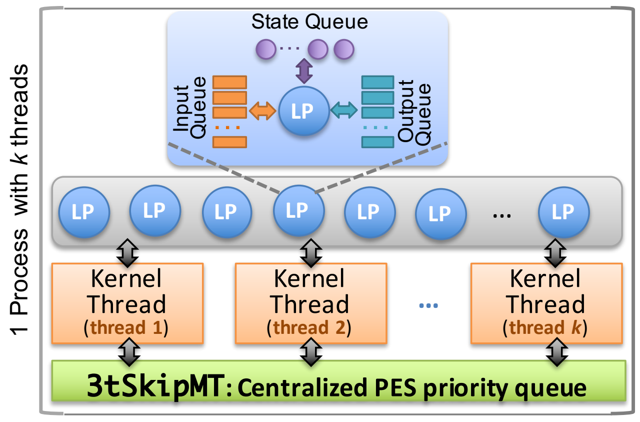


Figure 2: Overview of multithreaded MUSE PDES

Each simulation-kernel thread dequeues an LP with the next lowest timestamped event from `3tSkipMT` and process it. Since dequeue removes the LP and the operation is thread-safe, each thread acquires exclusive access to process an LP. Note that at any given time each thread operates on only one LP. However, each thread can concurrently enqueue future events into the `3tSkipMT` to any LP. Once an LP completes processing events, the thread enqueues the LP back and proceeds to process the next LP. This scheme provides an inherent load balancing scheme between the threads as opposed to fixed partitioning of LPs to threads. Moreover, it reduces the inherent time differences between LPs, thereby decreasing the probability of causal violations. Nevertheless, concurrent processing of events can cause rollbacks during PDES. The cancel operation on the `3tSkipMT` enables efficiently performing the necessary rollback operations. MUSE's LP API provides the necessary functionality for state saving and rollback-based causality restoration using suitable input and output event queues as shown in Figure 2.

Global Virtual Time (GVT) computation

In an optimistic PDES, states saved to enable rollback are garbage collected based on GVT. With our multithreaded simulations, we maintain a thread-local estimate of GVT. Periodically, the minimum of per-thread estimate is used to estimate GVT. This approach is straightforward and efficient, but the GVT can be underestimated, resulting in a slight increase in memory footprint. Each thread uses the GVT estimate to independently perform garbage collection in parallel.

Design for parallel garbage collection

In our proposed multithreaded PDES scheme, a single copy of an event is shared between sending and receiving LPs. Sending LPs maintain a reference for use during rollbacks. Consequently, garbage collection of events needs to be cognizant of concurrent use of events by two different LPs. Consequently, we have implemented a Dual Reference Counter (DRC) approach shown in Figure 3 to facilitate garbage collection. The sending and receiving LPs update two separate counters, thereby avoid race conditions. An event is deleted only when both counters are zero, indicating no thread is holding a reference to the event.

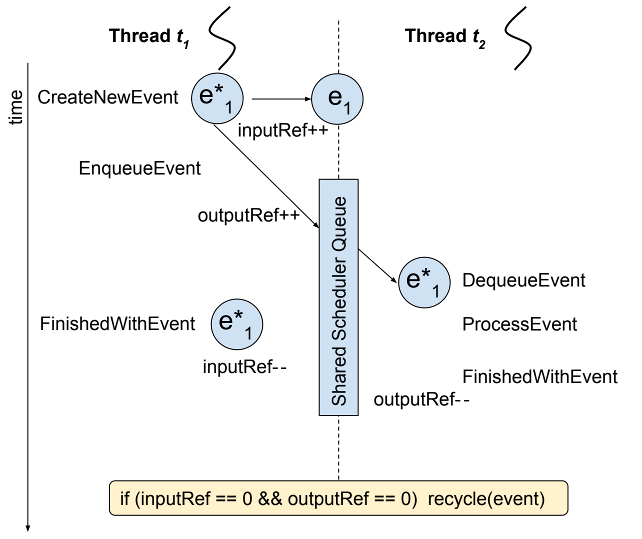


Figure 3: Dual Reference Counters approach used for concurrent garbage collection

DESIGN AND IMPLEMENTATION OF `3tSkipMT`

The data structure used to manage the Pending Event Set (PES) is called a scheduler-queue. It strongly influences the overall performance of a Time Warp synchronized PDES. As discussed in the BACKGROUND section, there are 4 key operations that are performed on a scheduler-queue, namely: enqueue, peek, dequeue, and cancel. These 4 operations have very different temporal, spatial, and concurrency characteristics. Consequently, as shown by our prior research (Rao and Higiro, 2019), an efficient approach for developing a scheduler-queue is to use a combination of different data structures organized into independent tiers. Specifically, inspried by the `3tHeap`, we propose a 3-tiered, concurrent data structure based on skip lists called `3tSkipMT`.

An overview of the three tiers constituting `3tSkipMT` is shown in Figure 4. The first tier is a priority queue of LPs, with priority being determined by the lowest timestamp event to be processed. That is, the LP with the earliest timestamp has the highest priority. This tier consists of a fixed number of LPs, but their order continuously changes as events are enqueued and dequeued in lower tiers. The second tier consists of a list of buckets, ordered based on the timestamp of events in a bucket. Each bucket contains a set of concurrent events, *i.e.,* events at the same timestamp, constituting the third tier. The third tier is implemented using a standard C++ `std::vector` while the other two tiers use a concurrent skip list is discussed next.

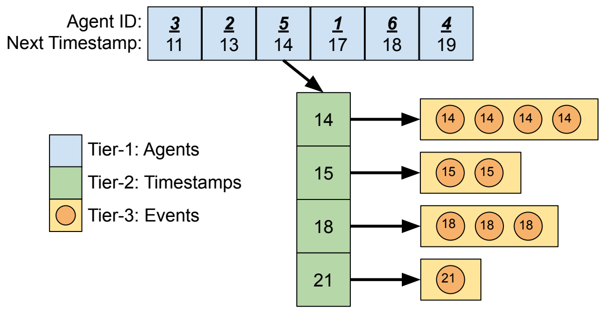


Figure 4: Overview of 3 tiers in `3tSkipMT`

Concurrent Skip List (for tiers 1 & 2)

In a multithreaded PDES, the first two tiers of the `3tSkipMT` will experience a significant number of concurrent operations, especially when event granularity is small. Consequently, we have adapted a skip list proposed by Lindén and Jonsson (2013) as the backing implementations for the first two tiers. This skip list has shown to have particularly good performance under very high contention conditions, yet are relatively simple to implement. By making balancing decisions for the search tree that are agnostic to the state of the queue, skip lists are particularly useful in concurrent structures where contention for shared state is the primary bottleneck on performance.

Figure 5 illustrates the overview of our Concurrent Skip List (CSL) adapted from Lindén and Jonsson (2013). Our CSL is designed to handle simultaneous insert and delete operations without data loss, duplication, or inaccuracy. There are two important regions of the CSL: ❶ the upper level `next[i]` (i > 0) skip links, are used to reduce search complexity when traversing the queue top-to-bottom and ❷ the bottom level `next[0]` represents the true priority order of events in the queue. The "height" of a level (l) for a single node is chosen probabilistically $(p(l))$ via a geometric distribution $p(l) = 0.5 \cdot p(l-1)$. Starting with a 100% probability of reaching the bottom level, for a given level (l), the likely hood of a node reaching the level above it is reduced by half. Hence, traversing links from the top down creates an effective binary search for long lists.

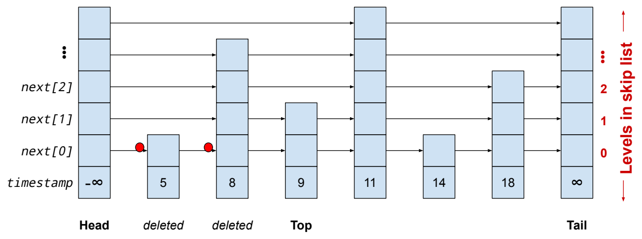


Figure 5: Concurrent Skip List (CSL) used for tiers 1 & 2 of `3tSkipMT`

Enqueue, Peek, & Dequeue operations

As a result of the CSL design, we can concurrently insert and delete while maintaining causality by manipulating pointers on level-0. Next, the higher levels are suitably restructured to decrease search time but without impacting lower levels. These operations are performed using the `next[]` pointer while allowing for the detection of conflicting inserts and deletes on the same node using atomic compare-and-swap (CAS) instructions. Specifically, if a thread attempts to insert a new node directly before a node that is also being modified (by another thread), or if a thread attempts to delete the same node as another thread, the CAS instruction will detect a conflict. Conflicts due to concurrent modifications cause operations on one of the threads to fail, prompting those threads to then retry the operation.

Cancel operations: Issues & implementation

A novel modification that has been introduced in our CSL is the ability to delete an arbitrary key from the middle of the queue. Deleting arbitrary keys is required for event cancellations associated with rollbacks in a Time Warp PDES. In addition, event cancellations also require re-prioritizing events and LPs in two different tiers of `3tSkipMT`. Accordingly, implementation of event cancellation has been accomplished by introducing a second atomic operation during event deletion – *i.e.,* in addition to flipping a deleted-bit on the `next[]` pointer, we also clear the event pointer contained in the node as part of the atomic CAS instruction. Flipping the deleted-bit is performed to detect conflicting dequeue operations and repeat them as part of lock-free operations. In addition, we

add the requirement that the thread must also set the value reference to `null`, thereby logically deleteing the node to successfully complete the operation.
This enhancement allows threads to *logically* delete an element from the middle of the queue. However, these nodes are not immediately deleted but are allowed to naturally propagate to the top of the CPQ where they are detected and deleted. Consequently, the memory footprint of the CPQ slightly increases due to these logically deleted nodes. On the other hand, this approach eliminates race conditions that arise when two different threads attempt to insert a new node immediately before the node being deleted. Importantly, it enables us to reduce contention and improve scalability as highlighted by our experimental results.

EXPERIMENTS & RESULTS

The primary objective of our research is to enable efficient multithreaded PDES. Accordingly, we have a widely used synthetic benchmark called PHOLD, discussed in the next subsection, for our experiments. Our initial set of experiments focused on verifying that our proposed pending event set queue `3tSkipMT` was operating correctly . The next set of experiments focused on exploring performance characteristics of MUSE which are discussed in the latter subsections. The last subsection presents some preliminary performance comparisons with another Time Warp based simulator called Ultimate Share-Everything (USE) by Ianni et al. (2018), to provide a more comprehensive perspective on the scalability of MUSE.

Experimental platform

The experiments were run on the Miami University Redhawk Supercomputing cluster. Each node of the cluster contains two Intel ® E5620 CPUs @ 2.4GHz providing a total of 8 cores (hyperthreading disabled) with 32 GB of RAM in Non-Uniform Memory Access (NUMA) configuration. Trials were individually run on a single dedicated compute node with up to 8 threads (1 thread per core), with the average of 3 runs being taken per data point.

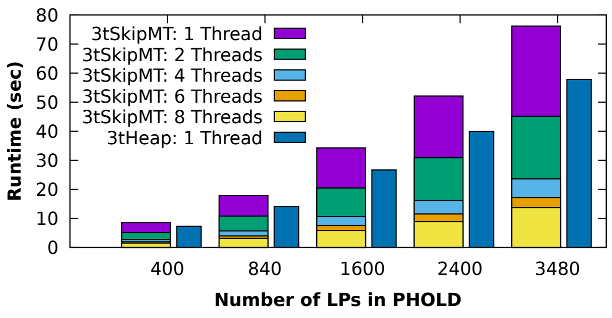
PHOLD synthetic benchmark

Proposed by Fujimoto (1990), PHOLD is widely used in PDES investigations because it has shown to effectively emulate the steady-state phase of a broad range of real world models (Rao and Higiro, 2019). A key advantage of PHOLD is that it includes a variety of settings that can be used to configure the benchmark to mimic characteristics of different real world models. The benchmark consists of a 2-dimensional toroidal grid of interacting Logical Processes (LPs). The dimensions of the torus can be varied to reflect models of different sizes.
In PHOLD, the LPs exchange events between each other such that the total number of events in the simulation remains a constant. An event's timestamp and destination LP are determined based on different distributions to characterize models from various application domains. For example, a `uniform` distribution is used to characterize events typically observed in digital logic simulations. A `poisson` distribution is used to reflect the properties of queuing systems and network models. An `exponential` distribution is commonly used to mimic processes in which events occur independently at a given mean rate.

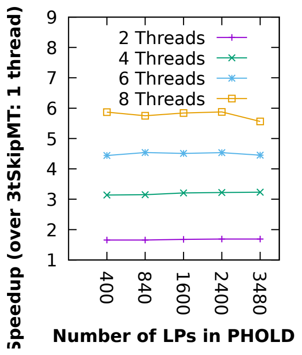
Event granularity
In real world models, event processing consumes CPU time, which is called the *granularity* of an event. In PHOLD, event granularity is characterized by performing a given number of trigonometric operations, that consume corresponding amount of CPU time. From our prior research (Rao and Higiro, 2019), we have used generalized sensitivity analysis and we have identified that only two parameters play an influential role this research, namely: ❶ the number of LPs which influences the net number of events in the simulation and ❷ each event's granularity. Accordingly, we explore these three influential parameters in the following subsection for performance assessments.

Scalability & performance results

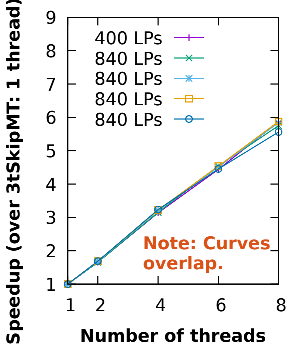
Recently, in Rao and Higiro (2019) we presented a novel single-threaded data structure called `3tHeap` that outperformed other state-of-the-art datastructures for PDES. That is, `3tHeap` is currently one of the fastest datastructures for PDES. Moreover, the 3-tier structure of our proposed `3tSkipMT` is comparable to that of `3tHeap`. Hence, in this section, we compare the performance of `3tHeap` to our new `3tSkipMT`. However, it must be noted that `3tHeap` is primarily designed to operate in a single-threaded mode. Consequently, we compare the scalability and performance of `3tSkipMT` with respect to the single-threaded performance of itself vs. `3tHeap`.
The chart in Figure 6(a) illustrates the observed average runtimes with a different number of threads and different number of Logical Processes (LPs) in the PHOLD benchmark. The chart also plots the average runtimes using the `3tHeap`, which is used as the reference for comparisons in this specific analysis. As illustrated by the chart in Figure 6(a), the performance of `3tSkipMT` is lower than the `3tHeap` by about 15% to 24%. This difference reflects two specific overheads of the `3tSkipMT`, namely: ❶ the overhead of using lock-free instructions which are known to degrade Instructions Per Clock-cycle (IPC) of the CPU (Rao, 2018), and ❷ slight increase in cache misses due to the use of linked lists in the `3tSkipMT`. However, as the number of threads is increased, the computation is spread across corresponding number of CPU-cores, thereby decreasing the net runtime of the simulations.
The plots in Figure 6(b) show the observed speedup realized from using multiple threads. As shown by the curves, as the number of threads are increased the performance of the simulation proportionally increases. However, the ob-



(a) Simulation runtime comparisons



(b) LPs vs. speedup



(c) Threads vs. speedup

Figure 6: Performance & scalability of `3tSkipMT`

served speedup is below the theoretical speedup because of contention between multiple threads that may need to operate on shared locations within the `3tSkipMT`. That is, when multiple threads operate on a shared location, the lock-free operations succeed only on 1 thread while other threads need to retry the operations, ensuring progress but impacting speedups. The plots in Figure 6(c) present the same data but from a scalability perspective. As illustrated by the relatively linear speedup curves, the `3tSkipMT` yields very good scalability characteristics as the number of threads are increased. Such scalability profile reflects the two key advantages of the design of `3tSkipMT` – *i.e.,* ① reduced contention within the data structure and ② the use of lock-free operations to minimize the computational overheads due to contention.

Influence of event granularity

In PHOLD, event granularity characterizes the compute time that an actual model would spend to process an event. Note that granularity is an artificer of a model and not the simulation-kernel or `3tSkipMT`. However, it is an important factor that indirectly influences the overall multithreaded performance as it determines contention for shared memory within the `3tSkipMT`. Large granularity values reflect scenarios where threads are operating independently for longer duration. On the other hand, small granularity values translate to frequent operations on the shared `3tSkipMT` from multiple threads which increases the chance of contention.

Figure 7 plots the impact of granularity on multithreaded simulation runtimes. The data was collected using a PHOLD benchmark with 625 LPs, initialized with 3 events-per-LP. The x-axis approximately (error $< 0.1\ \mu$s) corresponds to the event granularity. The slight approximation arises because we perform a given number of trigonometric operations and the actual CPU time taken to execute these operations varies slightly during runtime.

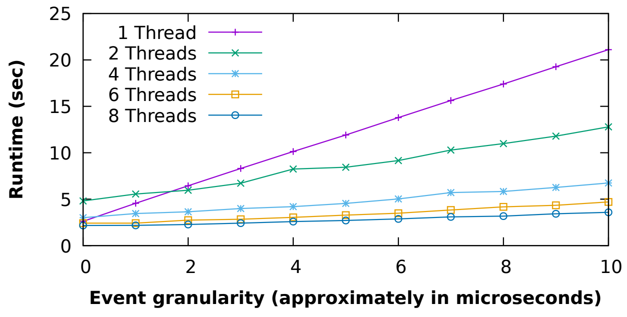


Figure 7: Influence of per-event granularity

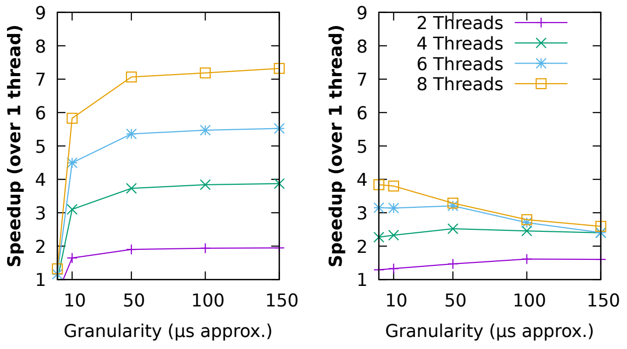
As illustrated by the curves in Figure 7, the `3tSkipMT` provides good multithreading performance as the event granularity increases. The bottlenecks due to increased contention are prominent only at zero granularity, where multithreaded simulations using 2, 4, and 6 threads are slower than the single-threaded one. The degradation in performance arises due to high contention between threads trying to operate concurrently and continuously on `3tSkipMT`. A zero granularity corresponds to a scenario where an actual model performs no additional processing, other than to schedule new events. In practice, however, every real world model performs some operations for each event. Consequently, a zero granularity scenario is purely a theoretical one used for empirical analyses. For all practical granularities, the data structure rapidly exhibits good performance highlighting the effectiveness of its design – *i.e.,* the design of `3tSkipMT` is effective at minimizing contention, thus mitigating the overheads by using lock-free operations.

Comparative scalability analysis

Scalability is a measure of how well a simulator is able to utilize added resources, *i.e.,* CPU cores. Scalability is important to enable efficient multithreaded simulations, particularly on many core CPUs. In order to determine the scalability of `3tSkipMT` in a larger context, we have compared its scalability to another simulator called Ultimate Share-Everything (USE) by Ianni et al. (2018). It uses a Conflict Resilient Calendar Queue (CRCQ) that is lock-free to enable multithreaded, Time Warp synchronized PDES. The design objectives of CRCQ are very similar to that of `3tSkipMT`. Moreover, USE has its own version of PHOLD benchmark, easing the design of experiments.

For comparative analysis, we have configured USE to mirror the same number of LPs, event distributions, and granularity parameters used by our benchmarks. However, it must be noted that USE is a completely different simulator when compared to our experimental testbed. Consequently, a direct comparison of the runtimes of two systems is not meaningful. Instead, we only compare the relative behaviors of the two systems from the perspective of scalability.

The charts in Figure 8 illustrate a comparison of the scalability of `3tSkipMT` versus CRCQ/USE for different granularity settings. In these experiments, `3tSkipMT` continues to exhibit good scalability as the number of threads are increased. For example, with 8 threads, `3tSkipMT` provides slightly over 7× speedup, at granularity > 50 μs. The scalability profile of `3tSkipMT` is substantially better when compared to the scalability profile observed for CRCQ in Figure 8(b). The improved scalability profile is of `3tSkipMT` highlights one of the key benefits this novel data structure for multithreaded PDES.



(a) `3tSkipMT` (b) CRCQ/USE

Figure 8: Influence of per-event granularity

CONCLUSIONS & FUTURE WORK

Current advances in hardware technologies are trending towards high-density compute nodes with many-core processors. Commodity CPUs with 64 cores are projected to be readily available in the next few years. The ongoing hardware trends have catalyzed research into enabling efficient, shared-memory Parallel Discrete Event Simulation (PDES) to accelerating scientific discoveries. Multithreading is the primary methodology used for realizing such parallel simulations. In practice, realizing the aforementioned advantages of multithreaded PDES is challenging due to a myriad of contention, consistency, and performance issues with multithreading. In our preliminary studies, accomplishing optimistic PDES using conventional lock-based data structures was a challenge and we experienced large overheads.

This paper presented our efforts to enable efficient, scalable optimistic PDES on shared-memory platforms. The paper presented the design and implementation of our multithreaded, Time Warp synchronized, general purpose parallel simulator called MUSE. The paper discussed the rationales underlying its design decisions. The cornerstone of our multithreaded approach is a concurrent priority queue called `3tSkipMT` that is used to manage the Pending Event Set (PES). It is a shared centralized queue that manages pending events for all LPs in a multithreaded PDES. The design permits a free thread to immediately schedule an LP to process the next set of events. This scheme provides an inherent load balancing scheme between the threads. Moreover, it reduces the inherent time differences between LPs, thereby decreasing the probability of causal violations.

The `3tSkipMT` has been designed to be thread-safe – that is, multiple threads can concurrently operate on it without causing race conditions. Moreover, `3tSkipMT` is designed to maximize concurrency. Maximizing concurrency without compromising thread safety has been facilitated using a 3-tier design. Each tier is designed to efficiently perform a given type of operation and to reduce contention between threads. Furthermore, the first two tiers have been implemented using a novel Concurrent Skip List (CSL) to minimize contention between threads. The skip list operations use contemporary lock-free operations to minimize contention overheads.

We have developed a novel method for logically deleting arbitrary keys from the middle of the CSL. Even though the event is tagged for garbage collection, the CLS-nodes are not immediately deleted. Instead, we allow these nodes to naturally propagate to the top of the CPQ where it is detected and eventually deleted. Consequently, the memory footprint slightly increases but ensures good scalability. In addition, we have designed a dual reference counter approach for GVT-based garbage collection. This approach enables sharing a single copy of an event between LPs, thereby reducing memory footprint. In addition, it permits threads to perform garbage collection in parallel.

We have extensively verified and validated our multithreaded parallel simulator using the PHOLD benchmark. In addition, we have used PHOLD for performance assessment of our simulator. The experiments show that our methodology yields almost linear scalability characteristic. We observed a very good ~7.5× runtime reduction using 8 CPU-cores, particularly for event granularity > 50 μs. In contrast, a recent state-of-the-art simulator was unable to match our scalability profile. Our experiments highlight the effectiveness of our proposed methodology for enabling efficient multithreaded optimistic PDES.

Future work

We are continuing our investigations to further improve the performance of MUSE. There are a few aspects of `3tSkipMT` that can be further enhanced. First, we are exploring the possibility of minimizing pointer usage to improve CPU-cache performance. Second, we are exploring algorithms for lock-free restructuring of the upper tiers for faster scheduling of LPs. We also plan to conduct experiments on higher density compute nodes with up to 28 CPU-cores. Currently, our research has focused on enabling multithreaded PDES on a single compute node. The next major step would be to further extend it to enable multithreaded PDES on a collection of compute nodes. Such an extension would enable hybrid simulations involving a combination of shared- and distributed-memory approaches. These aforementioned enhancements would enable realizing efficient PDES on the next generation of exascale supercomputers that are already on the horizon.

REFERENCES

Brown R., 1988. *Calendar Queues: A Fast 0(1) Priority Queue Implementation for the Simulation Event Set Problem*. *Commun ACM*, 31, no. 10, 1220–1227. ISSN 0001-0782. doi:10.1145/63039.63045.

Fujimoto R.M., 1990. *Performance of Time Warp under Synthetic Workloads*. In *Proceedings of SCS Multiconference on Distributed Simulation*. SCS, 1, 23–28.

Gupta S. and Wilsey P.A., 2014. *Lock-free Pending Event Set Management in Time Warp*. In *Proceedings of the ACM SIGSIM PADS*. ACM, New York, NY, USA. ISBN 978-1-4503-2794-7, 15–26.

Higiro J.; Gebre M.; and Rao D.M., 2017. *Multi-tier Priority Queues and 2-tier Ladder Queue for Managing Pending Events in Sequential and Optimistic Parallel Simulations*. In *Proceedings of the 2017 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. ACM, ACM, New York, NY, USA, SIGSIM-PADS '17. ISBN 978-1-4503-4489-0, 3–14. doi:10.1145/3064911.3064921.

Ianni M.; Marotta R.; Cingolani D.; Pellegrini A.; and Quaglia F., 2018. *The Ultimate Share-Everything PDES System*. In *Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. ACM, New York, NY, USA, SIGSIM-PADS'18. ISBN 978-1-4503-5092-1, 73–84. doi:10.1145/3200921.3200931.

Jafer S.; Liu Q.; and Wainer G., 2013. *Synchronization methods in parallel and distributed discrete-event simulation*. *Simulation Modelling Practice and Theory*, 30, 54–73. ISSN 1569-190X.

Lindén J. and Jonsson B., 2013. *A Skiplist-Based Concurrent Priority Queue with Minimal Memory Contention*. In R. Baldoni; N. Nisse; and M. van Steen (Eds.), *Principles of Distributed Systems*. Springer International Publishing, Cham. ISBN 978-3-319-03850-6, 206–220.

Marotta R.; Ianni M.; Pellegrini A.; and Quaglia F., 2017. *A Conflict-Resilient Lock-Free Calendar Queue for Scalable Share-Everything PDES Platforms*. In *Proceedings of the 2017 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. ACM, New York, NY, USA, SIGSIM-PADS '17. ISBN 978-1-4503-4489-0, 15–26. doi:10.1145/3064911.3064926.

Rao D.M., 2018. *Performance Comparison of Cross Memory Attach Capable MPI vs. Multithreaded Optimistic Parallel Simulations*. In *Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. ACM, New York, NY, USA, SIGSIM-PADS '18. ISBN 978-1-4503-5092-1, 37–48. doi:10.1145/3200921.3200935.

Rao D.M. and Higiro J.D., 2019. *Managing Pending Events in Sequential and Parallel Simulations Using Three-tier Heap and Two-tier Ladder Queue*. *ACM Trans Model Comput Simul*, 29, no. 2, 9:1–9:28. ISSN 1049-3301. doi:10.1145/3265750.

Tang W.T.; Goh R.S.M.; and Thng I.L.J., 2005. *Ladder Queue: An O(1) Priority Queue Structure for Large-scale Discrete Event Simulation*. *ACM Trans Model Comput Simul*, 15, no. 3, 175–204. ISSN 1049-3301.

Wang J.; Jagtap D.; Abu-Ghazaleh N.; and Ponomarev D., 2014. *Parallel Discrete Event Simulation for Multi-Core Systems: Analysis and Optimization*. *IEEE Transactions on Parallel and Distributed Systems*, 25, no. 6, 1574–1584. ISSN 1045-9219. doi:10.1109/TPDS.2013.193.