insert a nice title over here

Julius Higiro

A thesis proposal submitted for the degree of Master of Science in Computer Science

August 2017

1 Introduction

Discrete event simulation (DES) is the simulation of a discrete event system that contains states that operate at discrete points in time [1]. At each point in time in a simulation, a virtual timestamp is assigned to an event and the event precipitates a transition from one state to another state. This change in system state is used to represent the dynamic nature and behavior of a real-world system [2]. DES has been used in a variety of fields in academia, industry and the public sector as a tool to help inform our knowledge of discrete event systems and to improve decision-making processes [1]. DES provides an effective means for analyzing real or artificial systems without the constraint of limited resources such as time, financial costs, or safety. For example, the simulation of a battlefield environment can deliver insightful information to military planners on enemy troop movements, tactics and capabilities during strategic planning efforts [3]. A discrete event simulation of the battlefield allows military leaders to examine the impacts of decisions without the real-world risks associated with committing forces to dangerous environments.

Parallelism in computing frameworks that support DES increase performance throughput that is needed to construct and execute large scale and complex simulation models. With the growth and prevalence of semiconductor technology, cheaper and powerful multi-processors can be instrumented to achieve greater computing power for parallel discrete event simulations (PDES). However, the speedup achieved using multi-core and multi-processor systems requires efficient parallel programs.

Sequential and parallel DES are designed as a set of logical processes (LPs) that interact with each other by exchanging and processing timestamped events or messages [4]. Events that are yet to be processed are called "pending events". Pending events must be processed by LPs in priority order to maintain causality, with event priorities being determined by their timestamps. Consequently, data structures for managing and prioritizing pending events play a critical role in ensuring efficient sequential and parallel simulations [5, 6, 7, 8]. The effectiveness of data structures for event management is a conspicuous issue in larger simulations, where thousands or millions of events can be pending [9, 10]. Large pending event sets can arise when a model has many LPs or when each LP generates / processes many events. Overheads in managing pending events is magnified in fine grained simulations where the time taken to process an event is very short. Furthermore, the synchronization strategy used in PDES, Time Warp as described in section 3, can further impact the effectiveness of the data structure due to additional processing required for rollback-based recovery.

1.1 Motivations

Many investigations have explored the effectiveness of a wide variety of data structures for managing the pending event set as discussed in section 3. Among the various data structures, the Ladder Queue proposed by Tang et al [11] has show to be the most effective data structure for managing pending events [12, 8], particularly in sequential DES. Accordingly, we aim to replace the heap-based data structures used in our Time Warp synchronized parallel simulator with the Ladder Queue. Section XX discusses our Ladder Queue implementation and its fine-tuning. The Ladder Queue outperformed our multi-tier heap-based data structures in certain sequential simulations, consistent with observations by other investigators [8, 10]. However, as detailed in section XX, the Ladder Queue was substantially slower in two cases:

- High Concurrency: larger number of concurrent events (i.e. events with same timestamp) per LP.
- Time Warp synchronized parallel simulations conducted on a distributed memory computing cluster. Conversely, our multi-tier data structures performed well in parallel simulations.

To provide a good balance for both sequential and optimistic parallel simulations, we propose a significant change to the design of the Ladder Queue. Our revised data structure, discussed in section XX, is called 2-tier Ladder Queue (2tLadderQ). Various configurations of the standard PHOLD benchmark are used to assess the effectiveness of the multi-tier data structures vs. our fine-tuned implementation of the Ladder Queue. Results from our experiments discussed in section XX shows 2tLadderQ provides comparable performance in sequential simulations but outperforms the Ladder Queue in optimistic parallel simulations. Our 3-tier heap (3tHeap) outperforms our 2tLadderQ in high concurrency scenarios.

2 Miami University Simulation Environment (MUSE)

The implementation and assessment of the different data structures was conducted using our parallel simulation framework called MUSE. It was developed in C++ and uses the Message Passing Interface (MPI) library for parallel processing. MUSE uses Time Warp and standard state saving approach to accomplish optimistics synchronization of the LPs to maintain causality in event processing.

In a Time Warp based simulation such as MUSE, the simulation is organized as a set of LPs that interact with each other by exchanging virtual timestamped events. Each LP in a simulation maintains an input, output and state queue. The input queue is used to handle pending events that have yet to be processed. The output queue stores antimessages, which are events that are sent to other LPs to cancel out previously sent events or messages. The state queue stores the state of the LP at each discrete point in virtual simulation time. A Time Warp LP also maintains a local virtual time (LVT) tha.

2.1 Epxerimental Platform

3 Simulation Models

3.1 Parallel HOLD (PHOLD)

3.2 Personal Communication Service (PCS) Networks

3.3 Epidemiology

4 Scheduler Queues

5 Related Work

Our research explores multi-tier data structures for managing the pending event set in sequential and optimistic parallel simulations. Specifically, we compare effectiveness of the data structures against our fine-tuned version of the Ladder Queue [11] because it has shown to be very efficient for sequential DES. Tang et al [11] showed that ladder queue performed better than Sorted-discipline (SCQ) and Unsorted bucket discipline (UCQ) calendar queues at managing discrete events because it addressed performance shortcomings associated with resizing operations.

Recently, Franceschini et al [8] compared several priority-queue based pending event data structures to evaluate their performance in the context of sequential DEVS simulations. They found that Ladder Queue outperformed every other priority queue based pending event data structure such as Sorted List, Minimal List, Binary Heap, Splay Tree, and Calendar Queue. Tang et al [11] and Franceschini et al [8] both use the classic Hold benchmark simulation model used in this study and described in section XX.

In contrast to earlier work, rather than using a linked list based implementation, we propose alternative implementation using dynamically growing arrays (i.e. std::vector). Furthermore, we trigger *Bottom* to *Ladder* re-bucketing only if the *Bottom* has events at different timestamps to reduce inefficiencies to the Ladder Queue to enable its efficient use in optimistic parallel simulations.

Dickman et al [12] compare event list data structures that consisted of Splay Tree, STL Multiset and Ladder Queue. However, the focus of their paper was in developing a framework for handling pending event set data structure in shared memory PDES. A central component of their study was the identification of an appropriate data structure and design for the shared pending event set. Gupta et al [13] extended their implementation of Ladder Queue for shared memory Time Warp based simulation environment, so that it supports lock-free access to events in the shared pending event set. The modification involved the use of an unsorted lock-free queue in the underlying Ladder Queue structure. Marotta et al [14] have contributed to the study of pending event set data structures in threaded PDES through the design of the Non-Blocking Priority Queue (NBPQ) data structure. A pending event set data structure that is closely related to Calendar Queues with constant time performance.

In contrast to aforementioned efforts, our research focuses on distributed memory platforms in which each parallel process is single threaded. Consequently, our implementation does not involve thread synchronization issues. However, our 2-tier design has the ability to further reduce lock contention issues in multithreaded environments and could provide further performance boost. To the best of our knowledge, the Fibonnacci heap (fibHeap) and our 3-tier Heap (3tHeap) are unique data structures that have potential to be effective in simulations with high concurrency.

References

[1] G. Fishman, *Discrete-event simulation: modeling, programming, and analysis*. Springer Science & Business Media, 2013.

[2] R. M. Fujimoto, "Parallel discrete event simulation," *Communications of the ACM*, vol. 33, no. 10, pp. 30–53, 1990.

[3] R. R. Hill, J. O. Miller, and G. A. McIntyre, "Simulation analysis: applications of discrete event simulation modeling to military problems," in *Proceedings of the 33nd conference on Winter simulation*, pp. 780–788, IEEE Computer Society, 2001.

[4] S. Jafer, Q. Liu, and G. Wainer, "Synchronization methods in parallel and distributed discrete-event simulation," *Simulation Modelling Practice and Theory*, vol. 30, pp. 54–73, 2013.

[5] D. W. Jones, "An empirical comparison of priority-queue and event-set implementations," *Communications of the ACM*, vol. 29, no. 4, pp. 300–311, 1986.

[6] R. Rönngren and R. Ayani, "A comparative study of parallel and sequential priority queue algorithms," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 7, no. 2, pp. 157–209, 1997.

[7] R. Brown, "Calendar queues: a fast 0 (1) priority queue implementation for the simulation event set problem," *Communications of the ACM*, vol. 31, no. 10, pp. 1220–1227, 1988.

[8] R. Franceschini, P.-A. Bisgambiglia, and P. Bisgambiglia, "A comparative study of pending event set implementations for pdevs simulation," in *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, DEVS '15, (San Diego, CA, USA), pp. 77–84, Society for Computer Simulation International, 2015.

[9] C. D. Carothers and K. S. Perumalla, "On deciding between conservative and optimistic approaches on massively parallel platforms," in *Proceedings of the Winter Simulation Conference*, WSC '10, pp. 678–687, Winter Simulation Conference, 2010.

[10] J.-S. Yeom, A. Bhatele, K. Bisset, E. Bohm, A. Gupta, L. V. Kale, M. Marathe, D. S. Nikolopoulos, M. Schulz, and L. Wesolowski, "Overcoming the scalability challenges of epidemic simulations on blue waters," in *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pp. 755–764, IEEE, 2014.

[11] W. T. Tang, R. S. M. Goh, and I. L.-J. Thng, "Ladder queue: An o(1) priority queue structure for large-scale discrete event simulation," *ACM Trans. Model. Comput. Simul.*, vol. 15, pp. 175–204, July 2005.

[12] T. Dickman, S. Gupta, and P. A. Wilsey, "Event pool structures for pdes on many-core beowulf clusters," in *Proceedings of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, SIGSIM PADS '13, (New York, NY, USA), pp. 103–114, ACM, 2013.

[13] S. Gupta and P. A. Wilsey, "Lock-free pending event set management in time warp," in *Proceedings of the 2Nd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, SIGSIM PADS '14, (New York, NY, USA), pp. 15–26, ACM, 2014.

[14] R. Marotta, M. Ianni, A. Pellegrini, and F. Quaglia, "A non-blocking priority queue for the pending event set," in *Proceedings of the 9th EAI International Conference on Simulation Tools and Techniques*, SIMUTOOLS'16, (ICST, Brussels, Belgium, Belgium), pp. 46–55, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016.