Experiments with Transactional Memory for Event Pool Management

Joshua A. Hay
Dept of of Electrical Engineering and Computing Systems
Cincinnati, OH 45221-0030
hayjo813@gmail.com

Philip A. Wilsey
Dept of of Electrical Engineering and Computing Systems
Cincinnati, OH 45221-0030
wilseypa@gmail.com

ABSTRACT

Categories and Subject Descriptors

D.1.3 [**Programming Techniques**]: Concurrent Programming—*parallel programming, distributed programming* ; I.6.8 [**Simulation and Modeling**]: Types of Simulation—*parallel, distributed, discrete event*

General Terms

Algorithms, Performance

Keywords

Time Warp, pending event lists, multi-core, threads, transactional memory

1. INTRODUCTION

The remainder of this manuscript is organized as follows. Section 2 provides a brief review of related studies with transactional memory. Section 3 provides some background on Time Warp and transactional memory. Section 4 reviews the software architecture of the Time Warp simulation kernel (WARPED) that is used in this work and presents some data showing how contention for the shared pending event set negatively impacts performance. Section 6 presents the results of our experimental analysis. Finally,Section 7 presents some concluding remarks.

2. RELATED WORK

3. BACKGROUND

3.1 The Time Warp Mechanism

3.2 Transactional Memory

4. WARPED: A TIME WARP SIMULATION KERNEL

WARPED is a discrete event simulation kernel that implements the Time Warp synchronization protocol [4]. It was originally designed and optimized for executing parallel simulations on a Beowulf Cluster containing single core processors. It is highly configurable and incorporates many different sub-algorithms (*e.g.,* periodic checkpointing [3], and lazy, aggressive, and dynamic cancellation [8]) of the Time Warp mechanism [3]. Structurally, the Logical Processes (LPs) of a simulation are grouped together on each processing node where the LPs are scheduled according to a Least-Timestamp-First (LTSF) event scheduling policy. The node architecture reduces the Time Warp housekeeping functions such as GVT estimation, termination detection, and fossil collection into a set of common services for the entire population of LPs on that node. This architecture is similar to that reported in [1] and [7].

Most recently several attempts to build a threaded extension of WARPED have been pursued [5, 6]. These studies have produced a solution that works reasonably well for smaller multi-core processors. The overall design structure depicting the main pending event pool and the executing threads is shown in Figure 1. A threaded instance of WARPED contains a manager thread and one or more worker threads. The *manager thread* (labeled `M` in Figure 1) processes the Time Warp housekeeping functions and also processes the receipt and transmission of event messages exchanged with remote nodes in the cluster (local event insertion is performed by the worker threads). Additional details on the operation of the manager thread are available in [6]. The *worker threads* (depicted as `W0` and `Wn` in Figure 1) are responsible for dequeueing and executing pending events and generating new events accordingly. The pending event sets are organized into a two level structure as described below.

The pending event lists for each LP are maintained as independent sorted[1] lists that are independently locked. The lowest timestamped event from each LP event list is placed in a common LTSF pending event queue. The (locked) LTSF queue is sorted and used by the *worker threads* to schedule the next event for execution. After dequeuing and processing an event from the LTSF, each worker thread will then access the pending event set of the LP corresponding to the event just executed and remove the next least-timestamped event for insertion back into the LTSF queue. An abstract

[1]Although the prospect of using a partially sorted data structure such as calendar queues [2], lazy queues [9], or ladder queues [10] is possible.

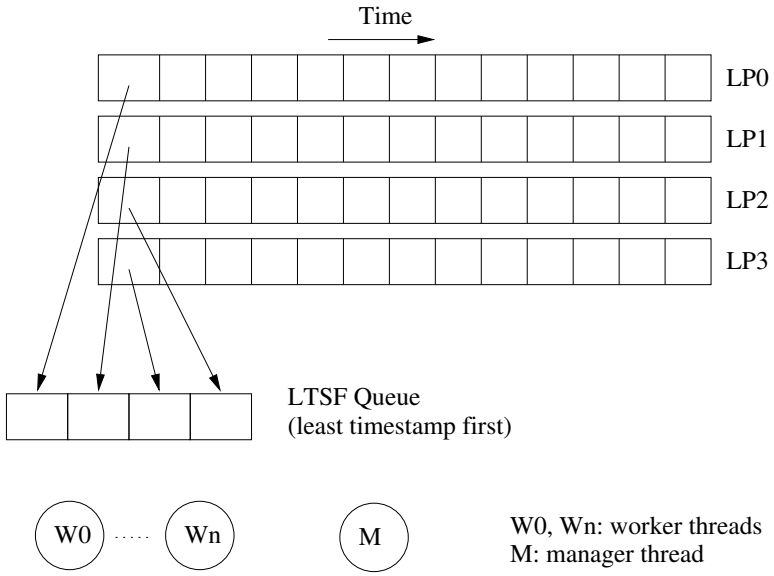

Figure 1: The principle pending event queues in warped.

```
worker_thread()

  lock LTSF queue
  dequeue smallest event from LTSF
  unlock LTSF queue

  while !done loop

    process event (assume from LPi)

    lock LPi queue

    dequeue smallest event from LPi

    lock LTSF queue

    insert event from LPi
    dequeue smallest event from LTSF

    unlock LTSF queue
    unlock LPi queue
  end loop
```

Figure 2: Generalized event execution loop for the worker threads. Many details have been omitted for the sake of clarity.

representation of the general event processing algorithm performed by the worker threads is shown in Figure 2.

While the above described design works well when the system is configured with only a few worker threads, once the number of worker threads exceeds 5-6, contention for the LTSF queue begins to negatively impact performance. Since the LP event pools are independently locked and since only one worker thread and the manager thread will simultaneously access the same LP event pool, contention to these structures is minimized. The principle point of contention for pending events in this architecture are at the LTSF queue. Thus, this study examines alternate designs for organizing the pending event list and especially the LTSF queue.

5. TRANSACTIONAL MEMORY IN HASWELL

6. EXPERIMENTAL ANALYSIS

7. CONCLUSIONS

8. ACKNOWLEDGMENTS

Support for this work was provided in part by the National Science Foundation under grant CNS–0915337.

9. REFERENCES

[1] H. Avril and C. Tropper. Clustered time warp and logic simulation. In *Proceedings of the Ninth Workshop on Parallel and Distributed Simulation (PADS'95)*, pages 112–119, June 1995.

[2] R. Brown. Calendar queues: A fast O(1) priority queue implementation for the simulation event set problem. *Communications of the ACM*, 31(10):1220–1227, Oct. 1988.

[3] R. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, Oct. 1990.

[4] D. E. Martin, T. J. McBrayer, and P. A. Wilsey. WARPED: A Time Warp simulation kernel for analysis and application development. In H. El-Rewini and B. D. Shriver, editors, *29th Hawaii International Conference on System Sciences (HICSS-29)*, volume Volume I, pages 383–386, Jan. 1996.

[5] R. Miller. Optimistic parallel discrete event simulation on a beowulf cluster of multi-core machines. Master's thesis, University of Cincinnati, 2010.

[6] K. Muthalagu. Threaded warped: An optimistic parallel discrete event simulator for clusters fo multi-core machines. Master's thesis, School of Electronic and Computing Systems, University of Cincinnati, Cincinnati, OH, Nov. 2012.

[7] R. Radhakrishnan, L. Moore, and P. A. Wilsey. External adjustment of runtime parameters in Time Warp synchronized parallel simulators. In *11th International Parallel Processing Symposium, (IPPS'97)*. IEEE Computer Society Press, Apr. 1997.

[8] R. Rajan and P. A. Wilsey. Dynamically switching between lazy and aggressive cancellation in a Time Warp parallel simulator. In *Proc. of the 28th Annual Simulation Symposium*, pages 22–30. IEEE Computer Society Press, Apr. 1995.

[9] R. Rönngren, J. Riboe, and R. Ayani. Lazy queue: An efficient implementation of the pending-event set. In *Proc. of the 24th Annual Simulation Symposium*, pages 194–204, Apr. 1991.

[10] 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 Transactions on Modeling and Computer Simulation*, 15(3):175–204, July 2005.