Trace-driven Simulation of Memory System Scheduling in Multithread Application

Pengfei Zhu[1,2] Mingyu Chen[1] Yungang Bao[1] Licheng Chen[1,2] Yongbing Huang[1,2]

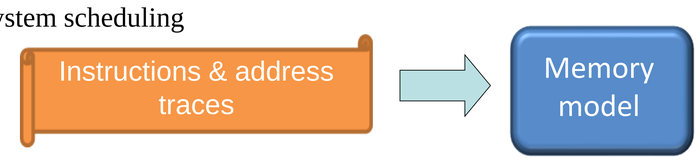
[1]Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
[2]Graduate University of Chinese Academy of Sciences, Beijing 100190, China

Introduction

Currently, trace-driven simulation is widely adopted in memory system scheduling research, since it is faster than execution-driven simulation and does not require data computation. On the contrary, due to the same reason, its trace replay for concurrent thread execution lacks data information and contains only addresses, so misplacement occurs in simulations when the trace of one thread runs ahead or behind others. This kind of distortion can cause remarkable errors during research. As shown in our experiment, trace misplacement causes an error rate of up to 10.22% in the metrics. This paper presents a methodology to avoid trace misplacement in trace-driven simulation and to ensure the accuracy of memory scheduling simulation in multithread applications, thus revealing a reliable means to study inter-thread actions in memory.

Motivation

- Trace-driven simulation of memory system scheduling
  - Less value computation
  - No data movement
  - Faster and more flexible



- How to handle inter-thread actions in multithread applications
  - Synchronization events: lock and barrier

- No data value in traces can be used to determine inter-thread behaviors when a memory scheduling simulator replays multithread application traces
  - Thread's actions in interleaved critical regions are hardly aligned with the trace collection process
  - Barriers do not take effect to force asynchronous threads to act as if they were synchronous.

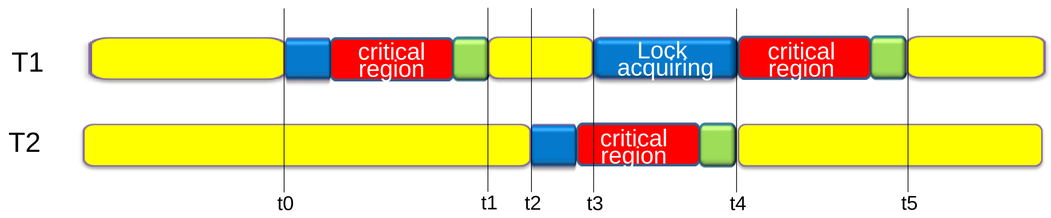


Figure 1 Trace alignment determined by lock

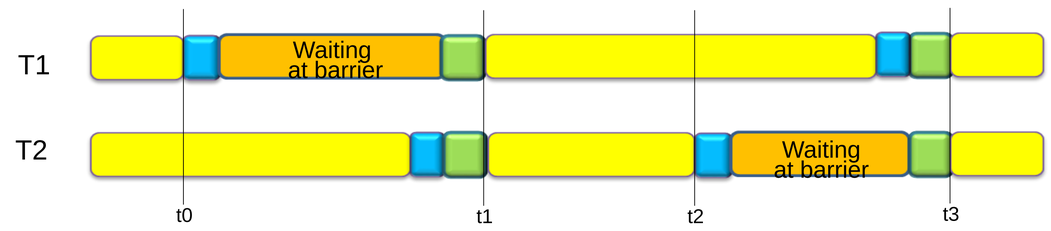


Figure 2 Trace alignment determined by barrier

- The trace misplacement
  - A misaligned lock acquiring order or an ineffective barrier causes a thread/core to run ahead or behind others in trace replaying.
  - The memory system evaluated with the proposed scheduling algorithm will be stimulated by misplaced conflict and interference in the shared resource
  - Misplaced multi-traces may reflect a particular disorder that may not even occur in the real system.

Methodology

- Targets of proposed methodology
  - to eliminate trace skew or trace misplacement in the simulation, as well as to inject an appropriate dynamic instruction trace into the simulation model according to the inter-thread action information

- Two components in the methodology
  - a set of instrumentation positions to recognize lock execution point and barrier execution point
  - a precise trace replaying method to maintain orders of synchronization events

Figure 3: Methodology conceptual scheme

- Two major steps from top to bottom:
  - The first step collects the workload's traces by the instrumentation, which particularly monitors the lock and barrier execution points in the dynamic instruction stream. The application is instrumented at the key positions to record the inter-thread ordering and synchronization, which are annotated in the trace.
  - The second step feeds the trace, including the ordering and synchronization annotation, into a memory scheduling simulation model, along with a set of pre-analyzed trace piece templates. The order and synchronization are faithfully maintained by the simulator. When the simulation reaches the annotated execution point, the simulator assembles the annotation into the trace piece template and injects the appropriate annotated trace pieces into the simulation according to the ordering requirement.

Instrumentation positions

- Thanks to the compilation technology, we can instrument additional functions at any workload execution points, statically or dynamically, to annotate information into traces:
  - critical region order at lock execution points
  - total number of threads waiting at barrier execution points
- Four instrumentation points for a lock and two points for a barrier
  - before-acquiring, after-acquiring, before-releasing, and after-releasing
  - before-entering and after-leaving.
- The instrumentations annotate inter-thread ordering sequence in trace files, and also filter out traces between positions of before-and-after pairs.

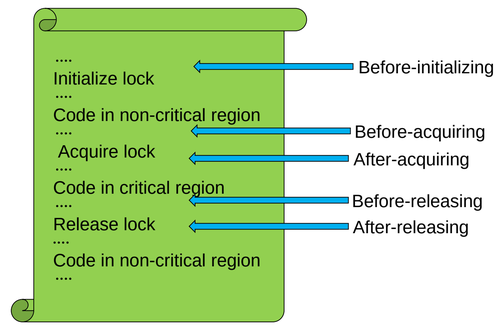


Figure 4 lock points to be instrumented

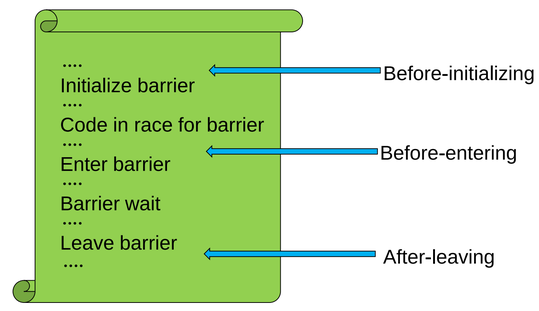


Figure 5 barrier points to be instrumented

Trace replaying

- To provide the precise order consistency with trace collection, trace-driven tool controls trace replay progress according to the order and synchronization tags annotated in traces, as well as the stimulus to the memory system.
- We propose trace-replaying simulation runs all traces at the beginning as a traditional trace-driven tool until any thread meets the annotated tags (i.e. lock acquiring, lock release, and barrier). Then the simulator compares the internal state with the tag to determine if the next trace piece from trace file will be run, instead of blindly running through the trace.
- Simulator cannot merely stop the instruction fetching from traces when the thread should be waiting for the lock, but loads the proper trace pieces, since the cache hierarchy (particularly the cache-coherent protocol) needs the stimulus filtered out at the collection stage. We propose reproducing trace from a trace piece template since traces at the lock and the barrier execution points are always uniform and repeated.

Evaluation

- Pin-based tool to collect traces and a cycle-accurate trace-driven tool to simulate memory scheduling algorithms.
- Replaying traces of PARSEC in memory scheduling simulator twice to compare metric results
  - First simulation: using proposed trace-replaying method
  - Second simulation: allowing disorder of critical regions and wrong synchronization of barriers.

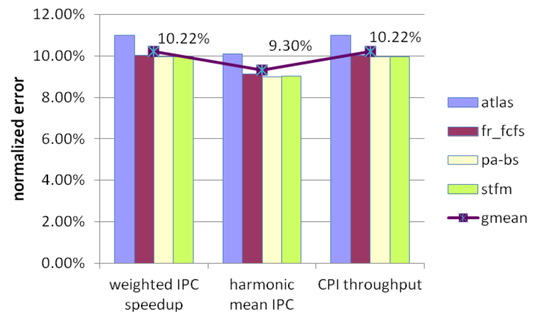


Figure 6 Normalized error between two simulations

- Each metric has a normalized error rate of nearly 10%. Putting memory scheduling algorithms together, the geometric mean error on weighted IPC speedup, harmonic mean of IPC, and CPI throughput are 10.22%, 9.30%, and 10.22%, respectively.
- The experiment shows that if we do not avoid the trace misplacement in the trace-driven simulation in multi-thread applications, the errors in the metric caused by trace-misplacement will be large enough to lead to erroneous conclusions, thus misleading the memory scheduling algorithm design.

Conclusion and future work

The methodology presented in this paper puts the emphasis on how to avoid trace misplacement in multithread applications when simulating the memory scheduling algorithm. We propose annotating the lock order and barrier synchronization in the trace to replay the deterministic inter-thread actions in the simulation. The described methodology provides a reliable method to simulate the memory scheduling algorithm in a multithread application.
In the future, more complex and important execution points will be analyzed, and the method to annotate behaviors of applications, such as dynamic or static load balance, from a higher level will be studied.

Acknowledgements

This work is partially supported by the National Science Foundation of China under grant numbers 61003062, 60903046, 60925009 and 60921002, and the National Basic Research Program of China (973 Program) under grant No. 2011CB302502.

中国科学院计算技术研究所
Institute of Computing Technology,Chinese Academy of Sciences