Trace-driven Simulation of Memory System Scheduling in Multithread Application

Pengfei Zhu, Mingyu Chen, Yungang Bao
Licheng Chen, Yongbing Huang

ICT, Chinese Academy of Sciences
MSPC-2012, Beijing

Outline

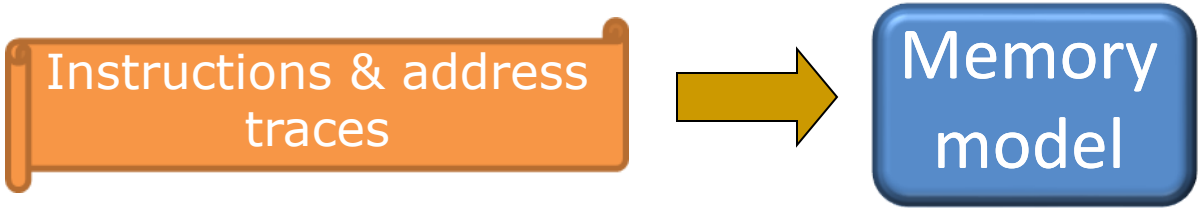
- Motivation
- Methodology
- Evaluation
- Conclusion and future work

Outline

- Motivation
- Methodology
- Evaluation
- Conclusion and future work

Motivation

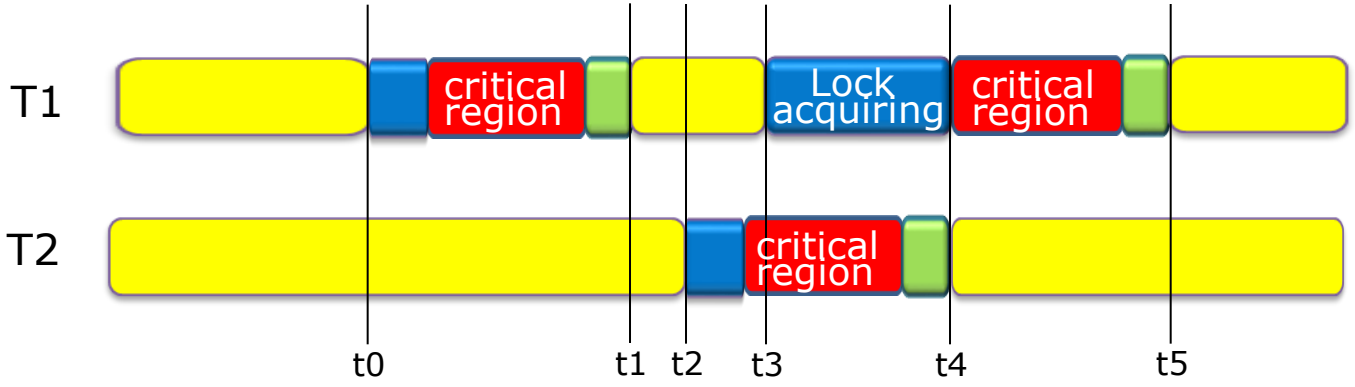
- Trace-driven simulation of Memory system scheduling
  - Less value computation
  - No data movement
  - Faster and more flexible



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

Locks in applications

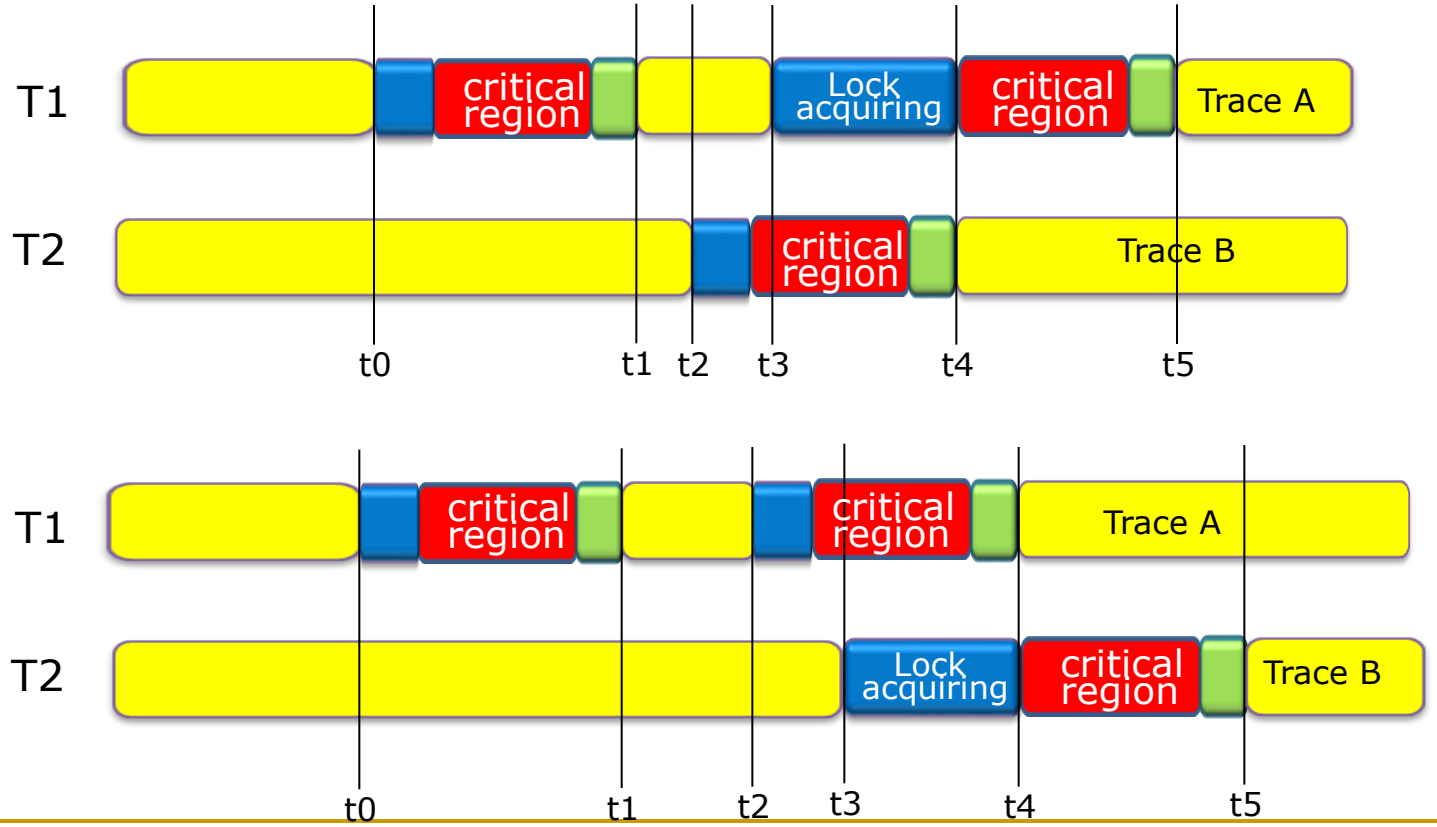
- lock order and inter-thread actions in trace collecting



- Replaying locks in trace
  - should maintain the critical region's mutual exclusion
  - But is it enough?

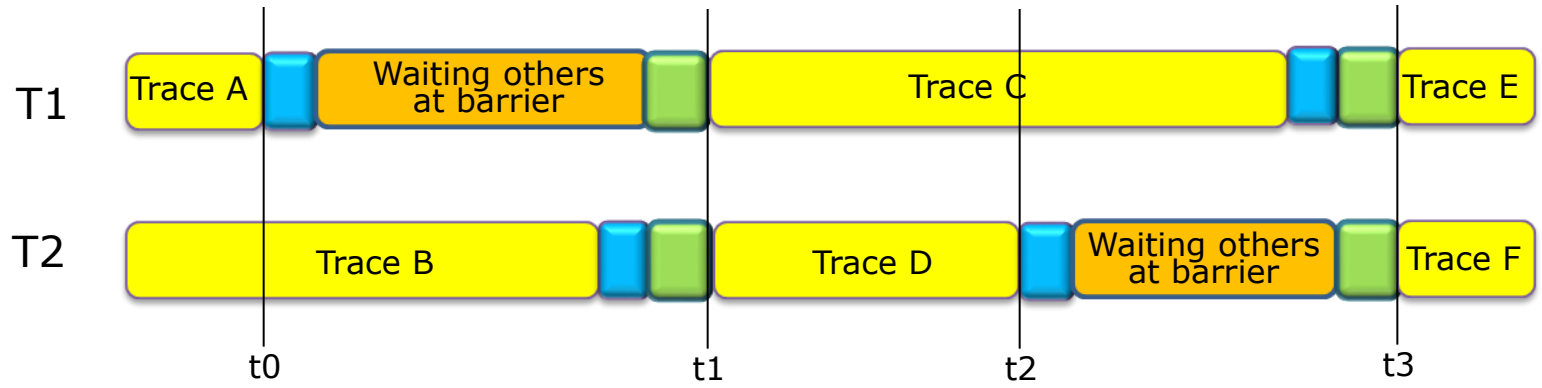
Trace misplacement

- Two different cases of trace replaying



Barriers in the simulation

- barrier behaviors in trace



- Replaying barriers in trace
  - is simpler than locks to avoid trace misplacement
  - should maintain an order between trace pieces on two sides of the barrier

Trace-driven problem in simulation

- memory accessing behaviors are determined by inter-thread actions
  - Particularly, producer-consumer relationship in multithread applications
  - Generally, multithread applications with dynamic load balance
- For given traces, if trace pieces orders are changed, trace misplacement causes incorrect memory inferences and conflicts to memory scheduler.
- Inter-thread actions are hardly maintained by simulation
  - no data in traces can be used to determine inter-thread actions collected by trace pieces

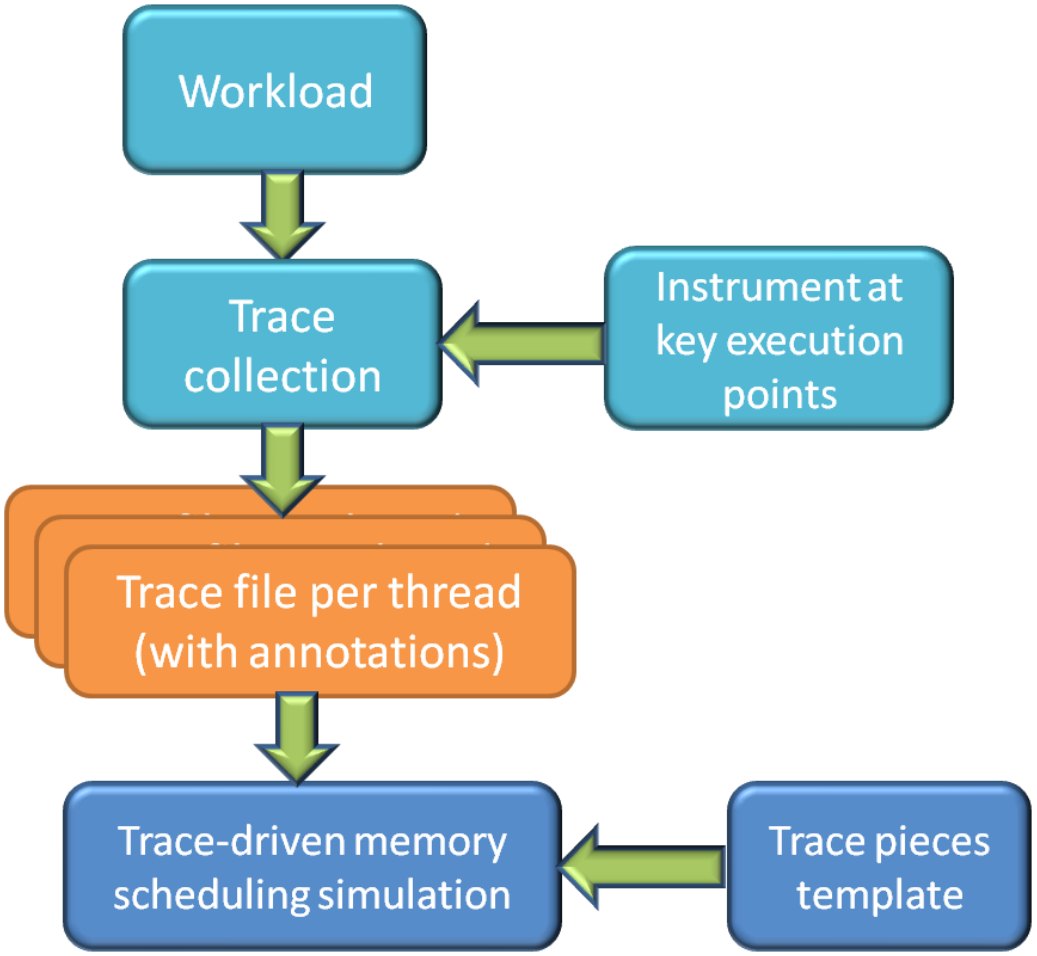
Outline

- Motivation
- Methodology
- Evaluation
- Conclusion and future work

Methodology

- To ensure trace pieces in the same order when collecting traces and replaying traces
- Two components in the methodology
  - a set of instrumentation positions
    - recognize two important execution points:
      lock execution point and barrier execution point
  - a precise trace replaying method in a trace-driven tool
    - Maintain orders of trace pieces

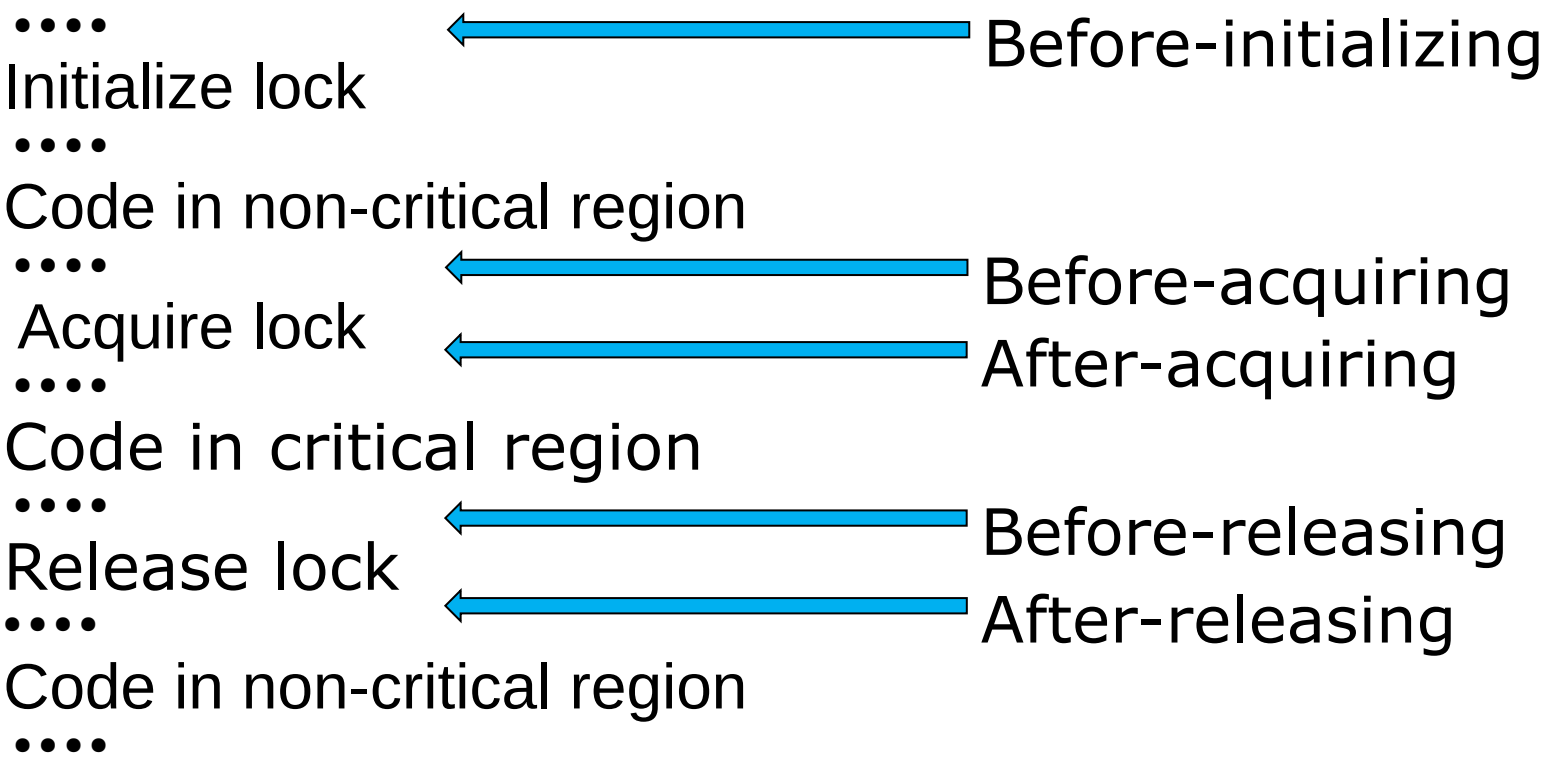
Conceptual scheme



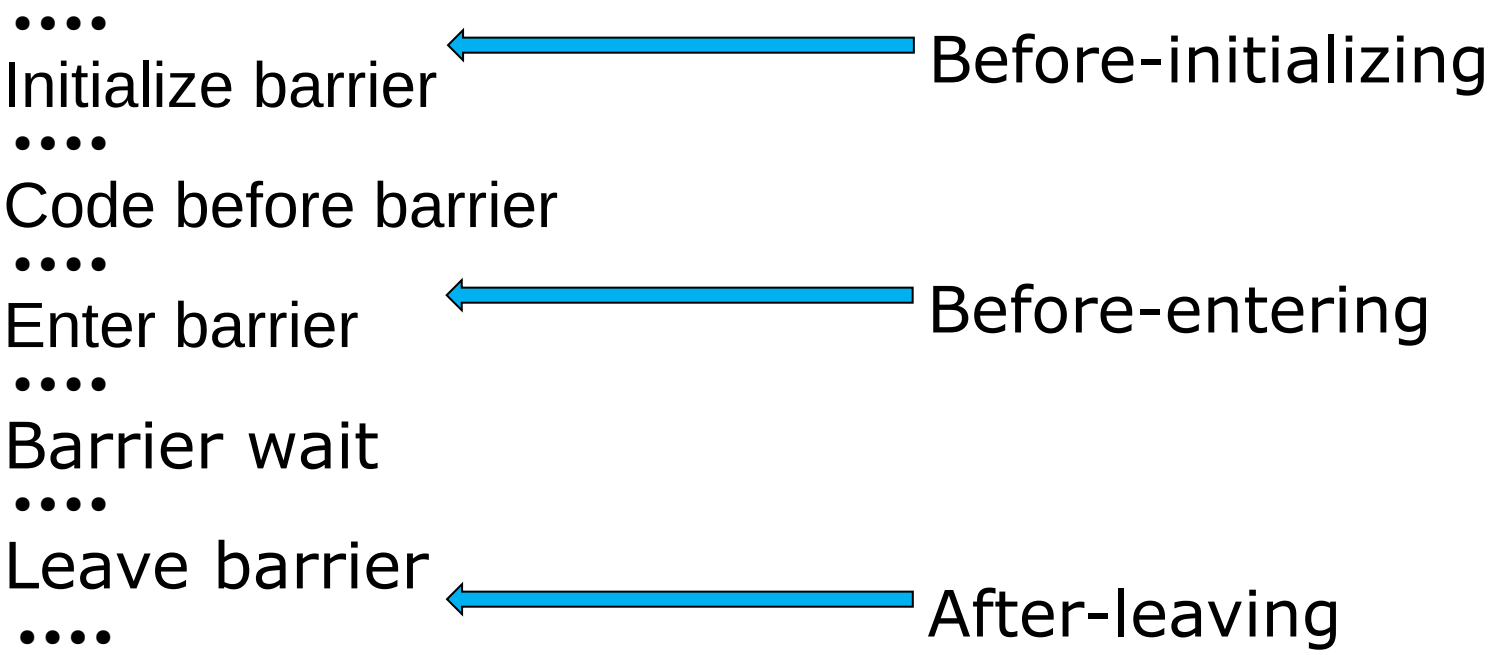
Instrumentation positions

- The goal is to collect the ordering and synchronization in the trace file
- Trace collecting relies on compilation technology
  - to instrument additional functions into workload, statically or dynamically.
- Through instrumentation, we annotate into traces
  - critical region order at lock execution points
  - total number of threads synchronized at barrier execution points

Lock execution points

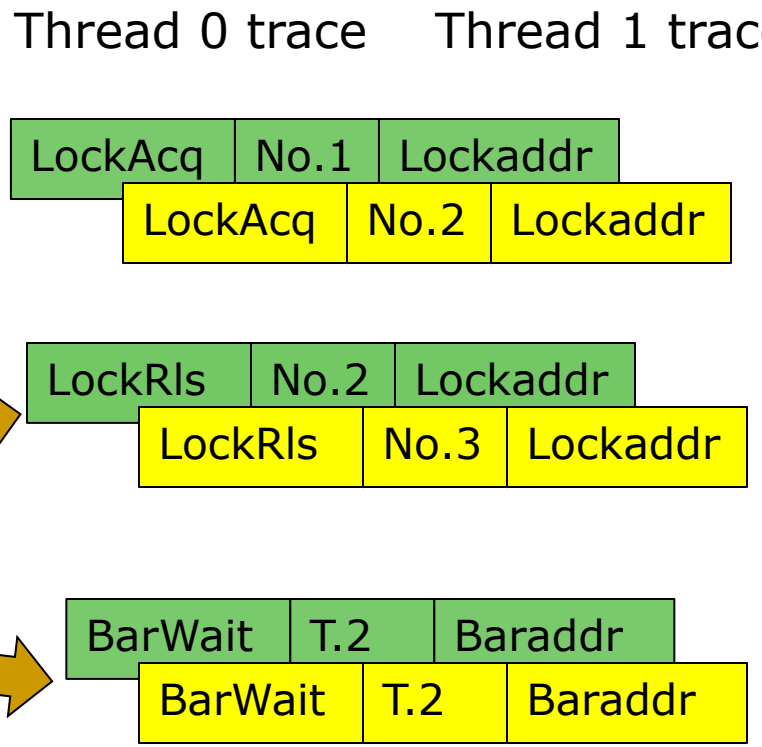


Barrier execution points



Trace with locks and barriers

```
int thread_func(…) {
    ….
    //code piece A
    pthread_spin_lock(&lock);
    //code in critical region
    pthread_spin_unlock(&lock);
    //code piece B
    pthread_barrier_wait(&wait);
    // code piece C
}
```



Trace replaying

- To provide the precise order consistency with the trace collection, the trace-driven simulator must control
  - the trace replay progress according to the lock order and barrier synchronization annotated in traces
  - the stimulus to the memory system.
    - Trace piece template

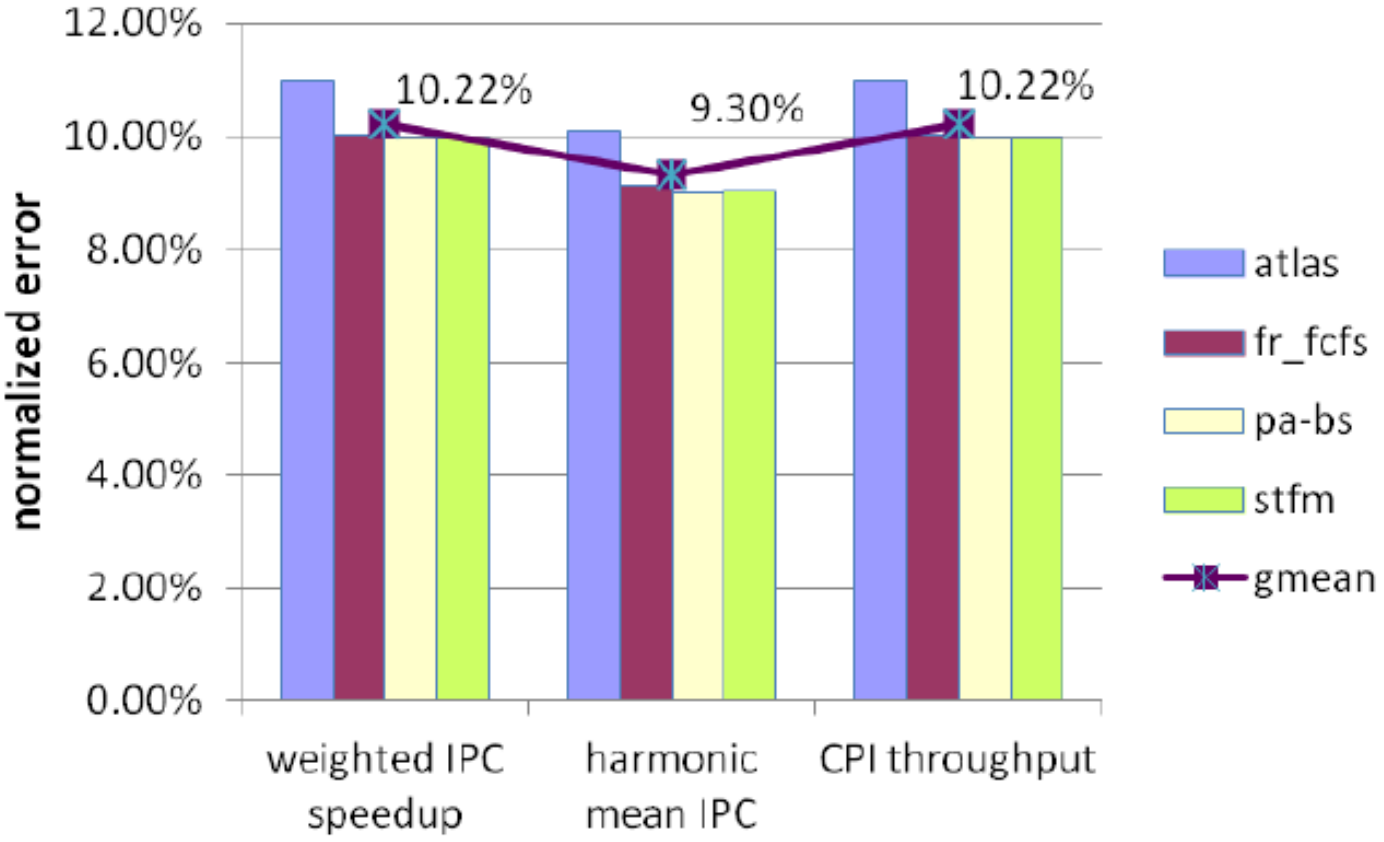
Outline

- Motivation
- Methodology
- Evaluation
- Conclusion and future work

Evaluation

- To prove the remarkable error caused by trace-misplacement
  - instrumentation function using Pin
  - a cycle-accurate trace-driven memory scheduling simulator
  - PARSEC benchmarks with locks and barriers
- Reply traces on memory scheduling simulator twice

Errors caused by trace-misplacement



Outline

- Motivation
- Methodology
- Evaluation
- Conclusion and future work

Conclusion

- Trace misplacement exists when simulating the memory scheduling algorithm in multithread applications
- Annotating lock order and barrier synchronization in traces and replaying deterministic inter-thread actions in simulation could avoid it.

Future work

- Methods annotating behaviors of applications from a higher level to differentiate the cases of trace piece orders
  - Deterministic replaying order
  - Non-deterministic replaying order
- More complex and important execution points

Thanks