APPLICATIONS OF THE ADAPTIVE TRANSACTIONAL MEMORY TEST PLATFORM

Mark Moir, Sun Microsystems Laboratories

Joint work with: Dave Dice, Maurice Herlihy, Doug Lea, Yossi Lev, Victor Luchangco, Wayne Mesard, Kevin Moore, Dan Nussbaum

Outline

- Background
  - TM, Rock
- Adaptive Transactional Memory Test Platform (ATMTP)
  - See Dan's talk later for details
- This talk:
  - overview of preliminary experiments using ATMTP
  - selected results, lessons learned

Background: Transactional Memory

- Transactional Memory (TM) is a promising technology for making it easier to develop concurrent programs that are scalable, efficient, and correct.
- Software (STM), hardware (HTM), or some combination (e.g., hybrid, hardware-assisted).
- “Unbounded” hardware solutions (a few years ago) incomplete, too complicated to consider for mainstream commercial processor.

Background: Rock

- Rock will support best-effort HTM
- Can abort transactions that exceed resources or encounter difficult events or instructions:
  - examples: function calls, `sdivx` instruction, exceptions
- Useful for improving performance of Hybrid TM
- Useful for a variety of other purposes too

HTM Instructions

- "Generic" best-effort HTM (as in ASPLOS 2006 paper)
  - `chkpt <fail_addr>`
  - `commit`
- Rock extension:
  - `rd %cps, <dest_reg>`
  - provides crucial feedback on reasons for transaction failure

ATMTP supports Rock HTM instructions

- First-order approximation of Rock's success/failure characteristics
- Goal: enable experimentation with HTM code before Rock is widely available.
- Goal is not to accurately simulate Rock implementation or performance
- Based on Wisconsins GEMS simulator, included in latest open-source release (GEMS 2.1). You can play too!

Preliminary exploration

- Overview of experiments described in paper:
  - Transactional red-black tree with HyTM and PhTM
  - Exposing TM to Java™ programs, optimising Java-based STM
  - DCAS-based collections in Java
  - Eliding locks explicitly for libc and STL
  - Eliding locks *implicitly* in Java
- Summary: some encouraging results, some less encouraging, some pitfalls identified and lessons learned

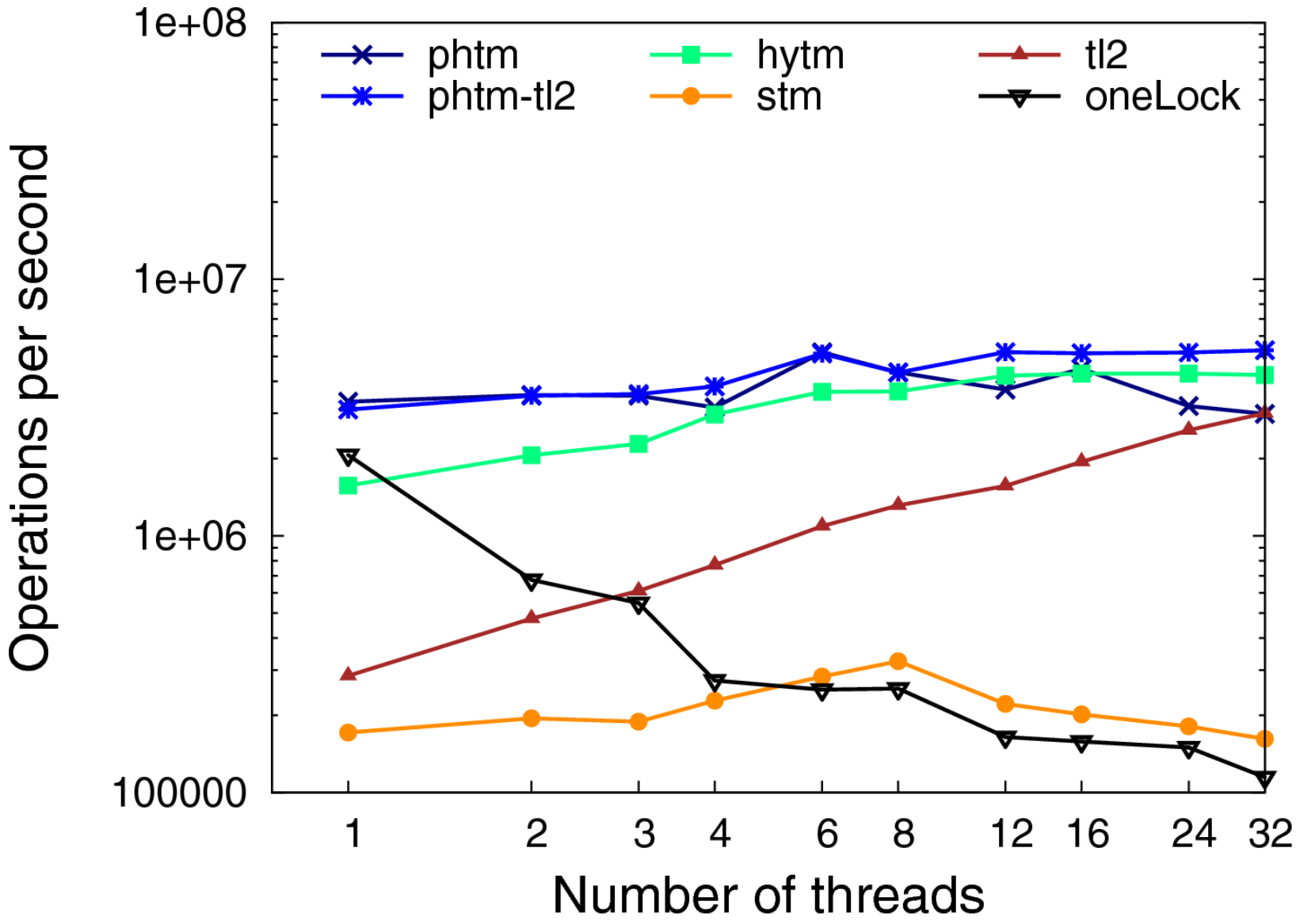
Preliminary exploration

- Overview of experiments described in paper:
  - Transactional red-black tree with HyTM and PhTM
  - Exposing TM to Java™ programs, optimising Java-based STM
  - DCAS-based collections in Java
  - Eliding locks explicitly for libc and STL
  - Eliding locks *implicitly* in Java
- Summary: some encouraging results, some less-encouraging, some pitfalls identified and lessons learned

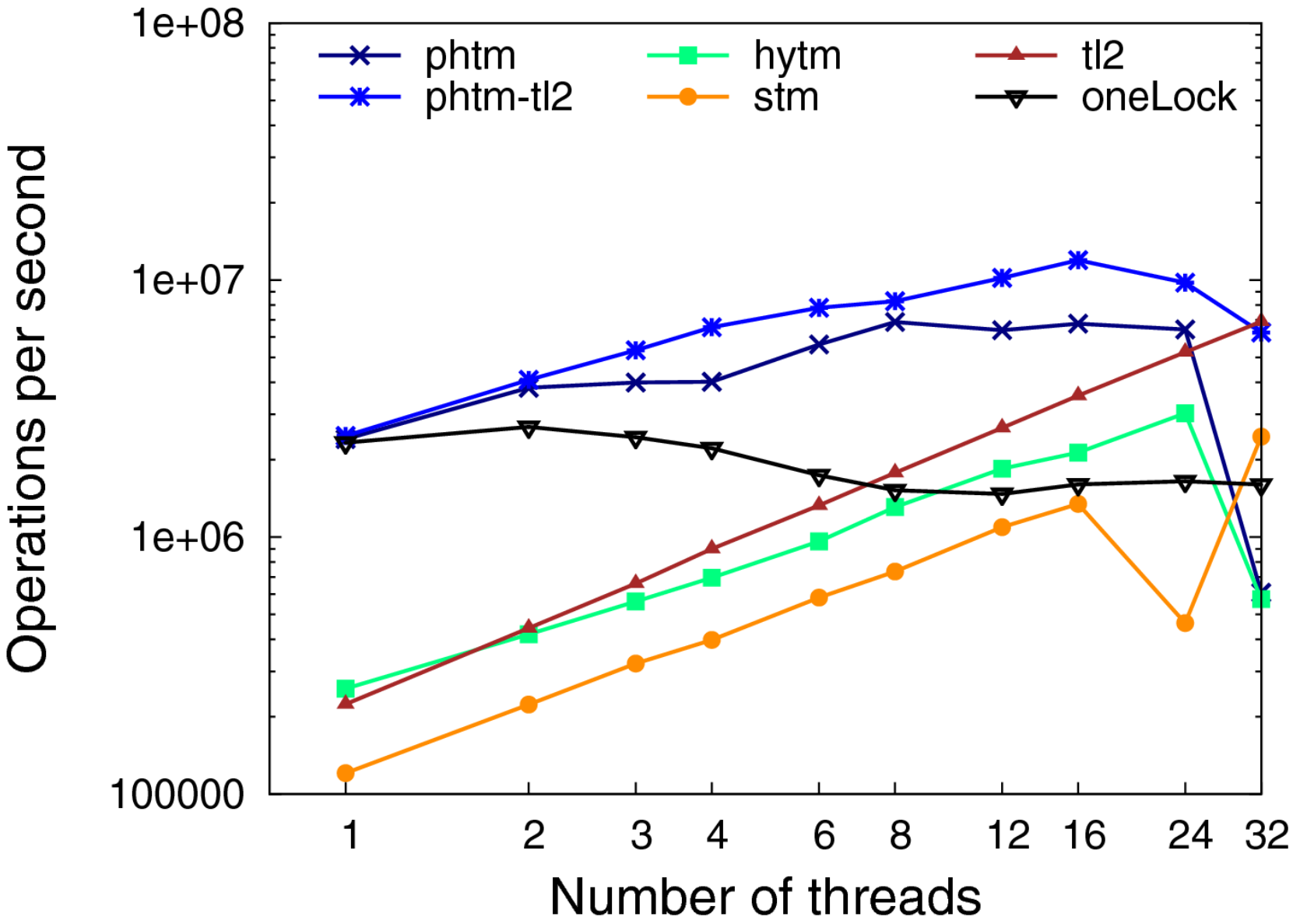
Transactional red-black tree

- Concurrent Red-Black trees very challenging
- We implemented transactional version using our HyTM compiler (ASPLOS 2006)
- Experiment:
  - keys in range [0,4095]
  - initialise tree with 2000 keys
  - each thread repeatedly inserts (20%), deletes (20%) or looks up (60%) randomly chosen key
  - measure total operations completed per second

Red-Black tree on old simulator



Red-Black tree on ATMTP



Red-Black tree with ATMTP

- Previous recursive version unsuccessful due to deeply nested function calls
- After switch to iterative version, inlining, PhTM successfully completes (almost) all operations using hardware transactions
- HyTM less successful: 30% of operations executed as software transactions
- Cause: TLB misses cause exceptions, txn is aborted so exception not processed, so TLB not loaded
- In this case, ITLB was the problem.
- Need “warm up” techniques to avoid repeated failure

DCAS-based collections in Java

- First DCAS approach (JDCAS):
  - Modify JVM to expose “unsafe” interface to HTM instructions
  - implement DCAS in Java inside transaction

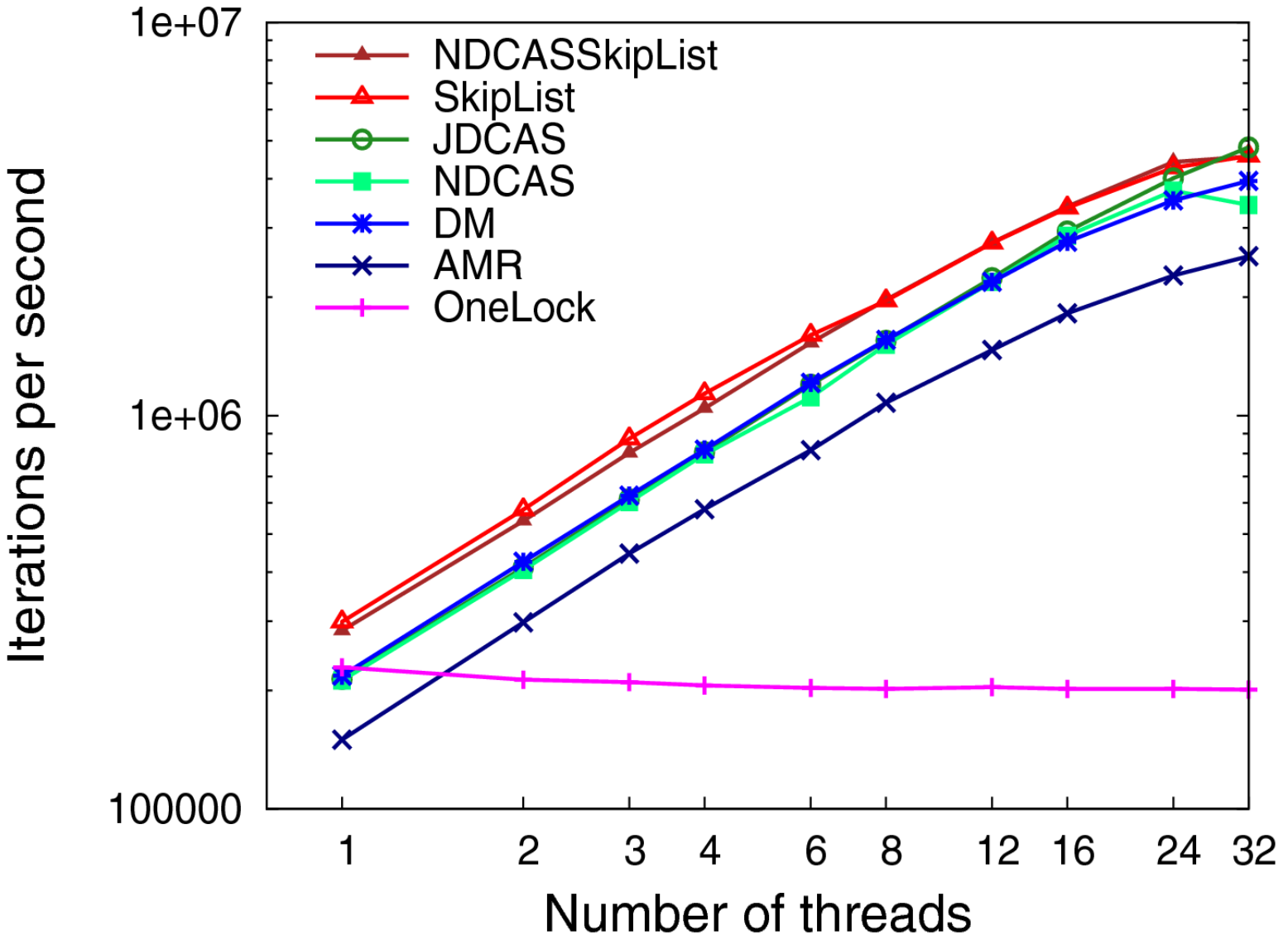
- Eventually worked well, after we learned some lessons:
  - compilation “catch 22”
  - data warmup tricky due to “hidden” code, e.g. class metadata needed for type cast
  - nonobvious conflicts due to (false) sharing on GC metadata; changes going into JVM to avoid this

DCAS-based collections in Java

- Second DCAS approach (NDCAS):
  - implement DCAS in native assembly code
  - no “surprise” code executed
  - factor GC metadata updates out of transactions
  - works well, but less flexible than general Java code inside hardware transaction

- Experiment:
  - key range [0..63]
  - each thread repeats: three lookups, one insert, one delete
  - measure total operations completed per second
  - compared existing list and skiplist from java.util.concurrent to new DCAS-based ones

DCAS-based collections in Java



Eliding synchronization in Java

- Modified JVM attempts synchronized blocks and methods using hardware transactions that:
  - start transaction
  - check lock is not held
  - execute critical section
  - attempt to commit; retry if unsuccessful
  - if repeatedly unsuccessful, eventually acquire lock

  (related to Rajwar and Goodman's “Speculative Lock Elision”, but hardware only provides atomicity, not decisions about whether/when to elide lock, whether/when to retry, etc.)

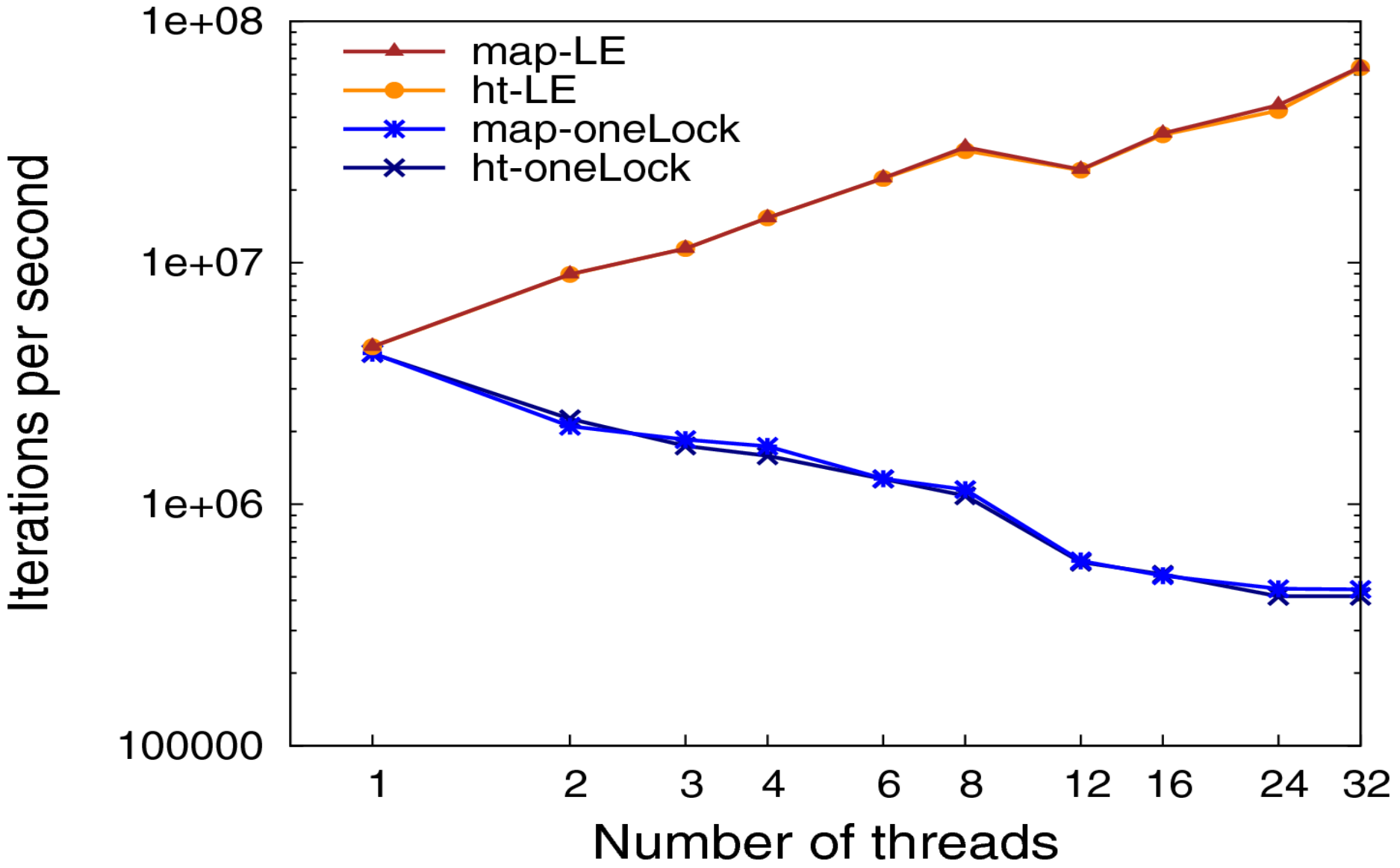
- Prototype does not yet revert to standard code when ineffective

Eliding synchronization in Java

- Experiment:
  - > Tested two collections from **`java.util`**:
    - **`HashMap`** (with synchronized wrapper)
    - **`HashTable`** (synchronization built in)
  - > Initialised collection with a set of objects
  - > Measured number of lookups of known-to-be-present objects per second

- **`HashTable`** was successful; operations all completed in hardware
- **`HashMap`** required small modification to factor division (sdivx instruction) ouf of transaction

Java lock elision



Concluding Remarks

- Best effort HTM is coming in Rock. Our simulator allows us and others to experiment with it before it arrives
- Preliminary explorations have yielded:
  - some encouraging results, some less encouraging
  - some pitfalls that require workarounds
  - some issues not easy to address or worth addressing before we have hardware and/or more accurate simulator
- ATMTP is a valuable tool for developing and testing HTM-based code before Rock is available; will also be useful afterwards

More information, papers, etc:
http://research.sun.com/scalable

Mark Moir

mark.moir@sun.com