



# HYBRID TRANSACTIONAL MEMORY

Mark Moir

Scalable Synchronization Research Group  
Sun Microsystems Laboratories

Joint work with: Peter Damron,  
Sasha Fedorova, Yossi Lev,  
Victor Luchangco, and Dan Nussbaum

# Outline

- Introduction
- Hybrid Transactional Memory
- Experiments and results
- War stories and lessons learned
- Conclusion

# Concurrent programming is hard today

- Coarse-grained locks don't scale.
- Fine-grained locks are difficult to manage (deadlock, lack of composition, etc.) and impose unnecessary overhead in uncontended case.
- Multicore is here, problem about to get much worse.
- Transactional programming can help.

# Example: queue transfer

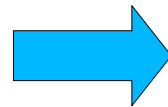
```
xferValue(Q1,Q2) {  
    Q1.lock.acquire();  
    Q2.lock.acquire();  
    v = Q1.dequeue();  
    Q2.enqueue(v);  
    Q2.lock.release();  
    Q1.lock.release();  
}
```

- Locks exposed for composition.
- Programmer must know and obey locking convention.
- This solution broken: deadlock.

# Transactional programming

- Express *what* to do atomically, not *how* to do it.

```
xferValue(Q1,Q2) {  
    Q1.lock.acquire();  
    Q2.lock.acquire();  
    v = Q1.dequeue();  
    Q2.enqueue(v);  
    Q2.lock.release();  
    Q1.lock.release();  
}
```



```
xferValue(Q1,Q2) {  
    atomic {  
        v = Q1.dequeue();  
        Q2.enqueue(v);  
    }  
}
```

# TM: a bit of history

- Hardware Transactional Memory (*HTM*):
  - > Herlihy and Moss (1993) (transaction size bounded by size of transactional cache).
  - > Recent resurgence in interest – *unbounded* HTM.
    - Hammond, *et al.* – TCC (2004).
    - Ananian, *et al.* – UTM (2005).
    - Rajwar, *et al.* – VTM (2005).
    - Moore, *et al.* – LogTM (2006).
    - ...

# TM: a bit of history (continued)

- Software Transactional Memory (*STM*):
  - > Shavit and Touitou (1995).
  - > Herlihy, *et al.* – DSTM (2003).
  - > Harris and Fraser – OSTM (2003).
  - > Saha, *et al.* – McRT-STM (2006).
  - > Dice, *et al.* – TL2 (2006).
  - > ...

# The dilemma

- Despite substantial progress, STM still slower than we would like.
- HTM expected to be much faster, but:
- Catch-22:
  - > Hardware guys: where is the software?
  - > Software guys: where is the hardware?

# Outline

- Introduction
- **Hybrid Transactional Memory**
- Experiments and results
- War stories and lessons learned
- Conclusion

# Hybrid Transactional Memory (HyTM)

- Run transactional programs on existing systems today.
  - > Encourage programmers to develop applications.
- Exploit future **best-effort** HTM to improve performance, with no additional programming effort.
- Relieve hardware designers of the burden of handling unbounded transactions, every corner case, all functionality.
  - > Minimal assumptions.
  - > Encourage them to implement *some* HTM support.
  - > Facilitates an incremental approach.
- The idea is to break the catch-22!

# HyTM: the basic idea

- Familiar design philosophy:
  - > Make the common case fast.
  - > Make the uncommon case work.
- Use HTM for (hopefully) common case.
- Use *compatible* STM otherwise.

# Prototype

- Word-based STM: suitable for C/C++.
- Compiler.
  - > HTM/STM code paths plus glue logic.
- Library.
  - > STM.
  - > Hooks for HTM/STM interaction.
- This is a *prototype*.
  - > Just one design point.
  - > Limitations.
  - > *Plenty* of room for improvement.
- Exists and works.

# The challenge

- HTM and STM transactions must interoperate.
- What if a hardware transaction conflicts with a software one?
- HTM *hardware* is not aware of software transactions.
- Make hardware *transactions* aware of software transactions.

# Our approach: library + compiler

- HTM path:

```

Y = X + 5;      ==>
                 txn_begin abortHandler;
                 if (!canHardwareRead (&X))
                   txn_abort;
                 tmp = X;
                 if (!canHardwareWrite (&Y))
                   txn_abort;
                 Y = tmp + 5;
                 txn_end;

```

# Making hardware transactions aware of software transactions

- Basic idea:
  - > Software transactions maintain data structures for read/write ownership.
  - > `canHardwareRead()` and `canHardwareWrite()` look at data structures.
    - If incompatible software transaction is detected, abort hardware transaction.
    - If software status changes, hardware transaction will abort on its own.

# Outline

- Introduction
- Hybrid Transactional Memory
- **Experiments and results**
- War stories and lessons learned
- Conclusion

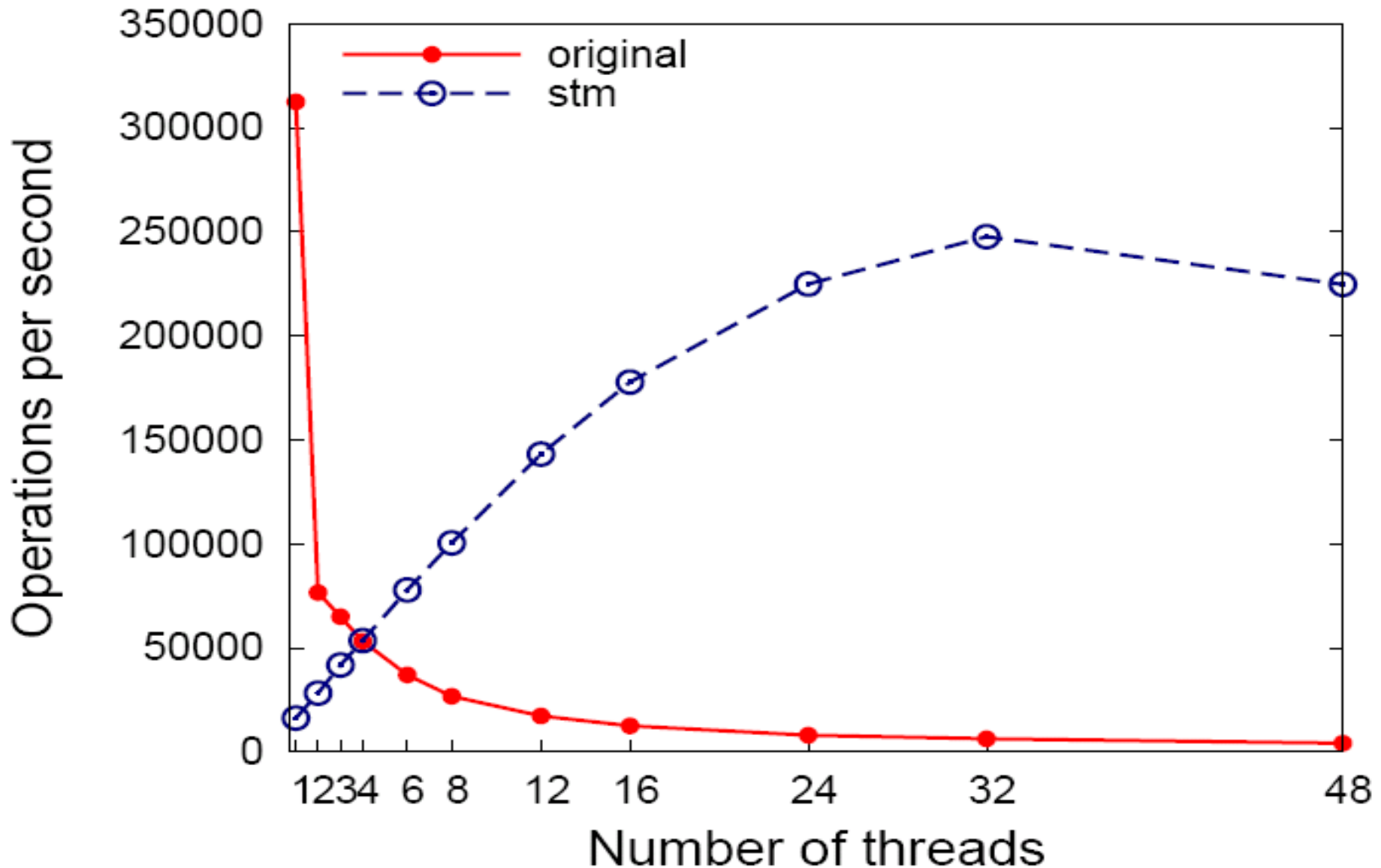
# Platforms

- 48-way Sun Fire™ 6800
- Simulator
  - > Starting point: Simics + Wisconsin GEMS + Wisconsin LogTM.
  - > Wired instructions produced by our compiler to LogTM instructions.
  - > Added support for failing to explicit abort handler.

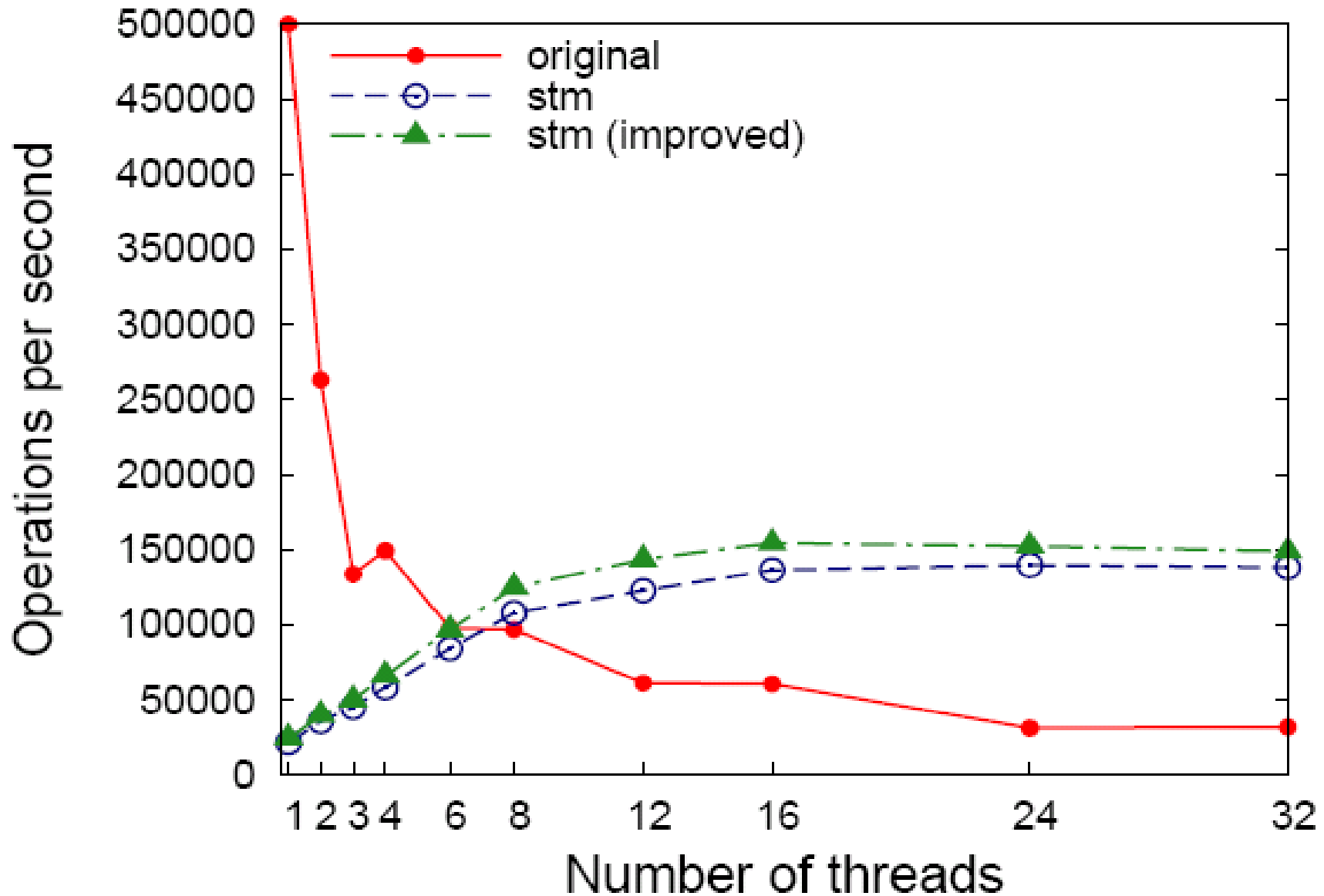
# Berkeley DB lock subsystem

- Manage read/write locking for client.
  - > Client supplies object to be locked and access mode.
  - > System allocates and manages locks.
- Production implementation uses one big lock to protect internal data structures.
- Benchmark: each thread repeatedly locks and releases a different object.
  - > Should scale well (in theory).

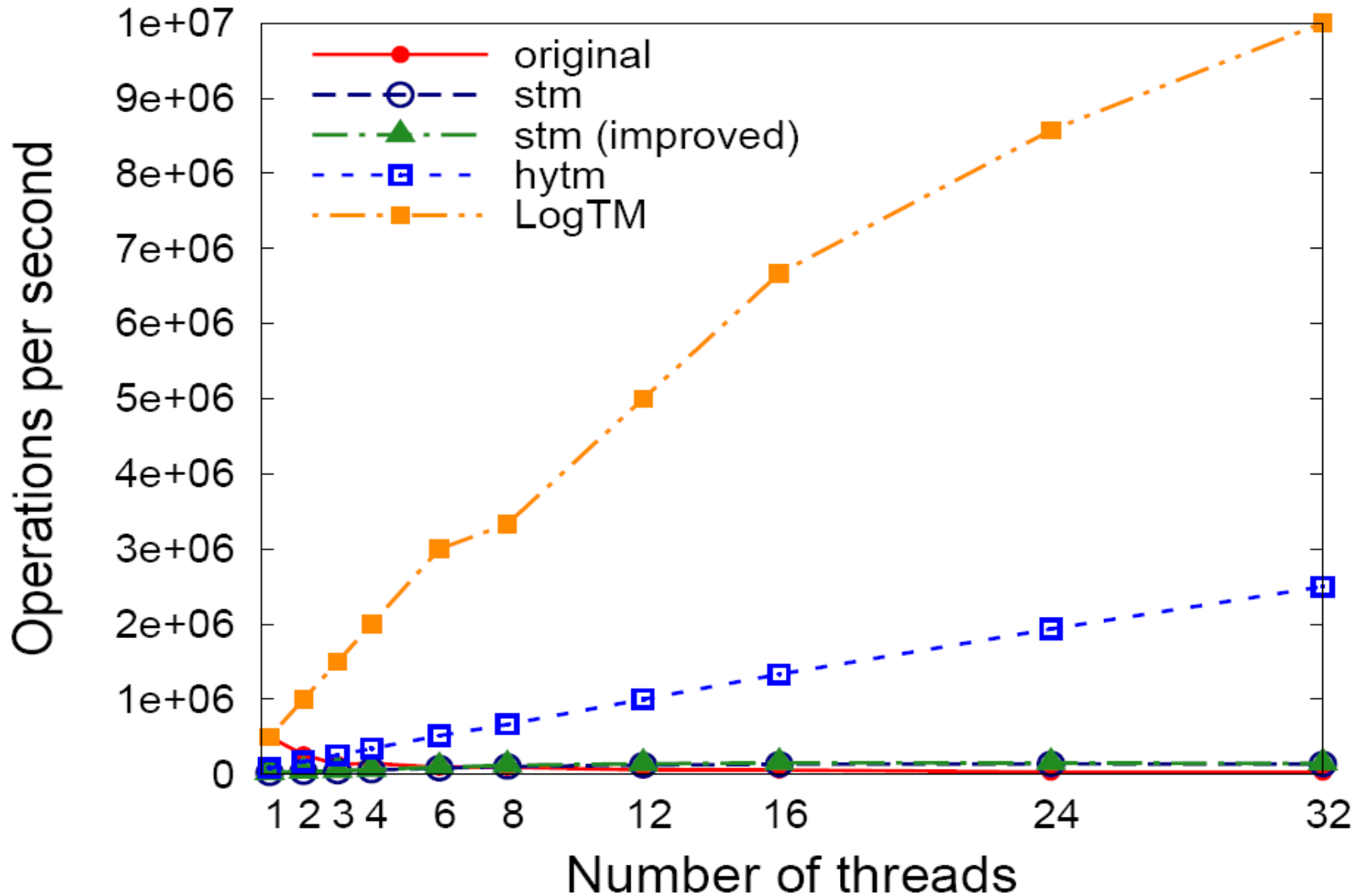
# Berkeley DB lock subsystem (STM only)



# Berkeley DB lock subsystem (simulation)



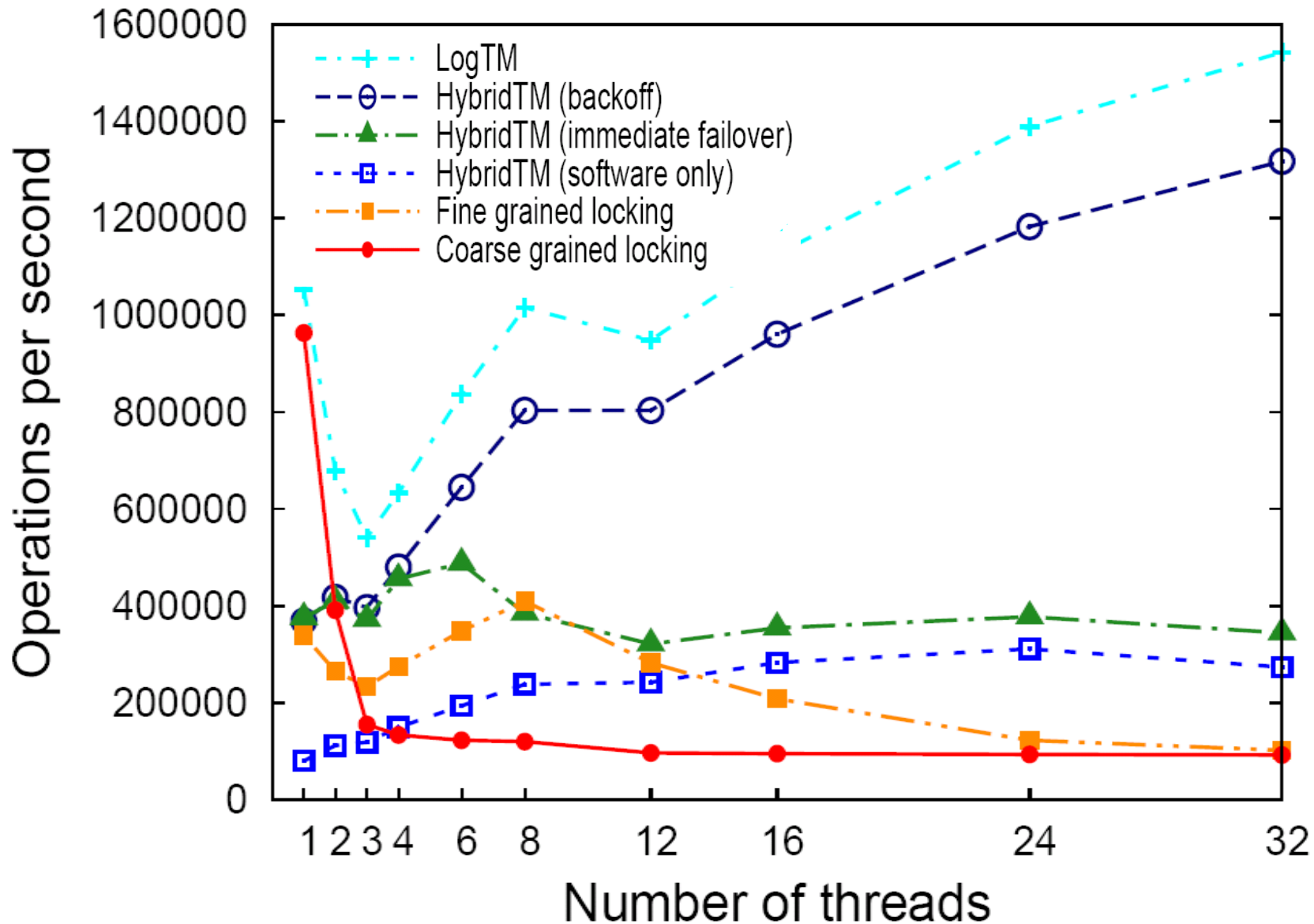
# Berkeley DB lock subsystem (simulation)



# rand-array benchmark

- Array of  $M$  counters.
- Each operation chooses  $K$  counters and increments them.
- In our experiments:
  - >  $K=10$
  - > For “low contention”,  $M=1,000,000$
  - > For “high contention”,  $M=1,000$
- Compared throughput using:
  - > LogTM
  - > HyTM in various configurations
  - > Coarse- and fine-grained locking

# Rand-array: high contention



# Summary

- Many pitfalls for scalability, can usually improve easily.
- STM:
  - > Scales better than coarse-grained lock, and sometimes even fine-grained locking under contention.
- HyTM:
  - > Can get performance and scalability comparable with unbounded LogTM, with much simpler HTM support.
- Room for improvement:
  - > Contention management.
  - > Overhead (especially single-threaded).

# Outline

- Introduction
- Hybrid Transactional Memory
- Experiments and results
- **War stories and lessons learned**
- Conclusion

# War stories and lessons learned

- False sharing: problem worse than usual, remedy the same.
- Real conflicts: easier to eliminate/reduce with transactions.
- Design choices and optimisations are workload dependent.
- Need flexible mechanisms and adaptive policies (harder?).
- Contention control: lots of room for improvement, but better to avoid contention.

# Message to hardware people

- Give us your best effort, don't give up if you can't do unbounded transactions with all possible functionality.
- Best-effort HTM supports other purposes too!
- To help you, we *assume* very little, but some things beyond basic assumptions can help us substantially, e.g., feedback.

# Message to software people

- Transactions can improve scalability and simplify your life substantially.
- OK, so we're not there yet (performance, functionality), but don't wait:
  - > Influence programming models and implementation.
  - > Motivate hardware people and provide guidance.

# Conclusion

- HyTM is a promising route to TM adoption.
- Breaks the catch-22:
  - > Can write simpler software today, get significant performance improvements as best-effort HTM appears and improves.
  - > Hardware people get more flexibility, more guidance, and more motivation.
- Plenty of work to do yet, but let's get started!



# Questions?

Mark Moir

[mark.moir@sun.com](mailto:mark.moir@sun.com)