

# An Efficient Lock-Aware Transactional Memory Implementation

ICOOOLPS 2009

Genoa, Italy – July 6, 2009



**Raytheon**

**Justin E. Gottschlich**

**University of Colorado at Boulder**

**Raytheon Company**

# Collaborators

---



**Daniel A. Connors**

**Maurice Herlihy**



**Jeremy G. Siek**



**Manish Vachharajani**



**Dwight Y. Winkler**

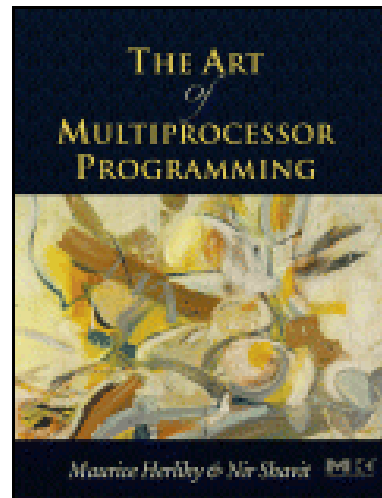


# The Problem:

Locks + TM  $\neq$  Correct Code

# Why Care About TM + Locks?

- Locks are prevalent in parallel software (Herlihy/Shavit)



- To be practical, TM must work with locks (Dice, PPOPP'07)

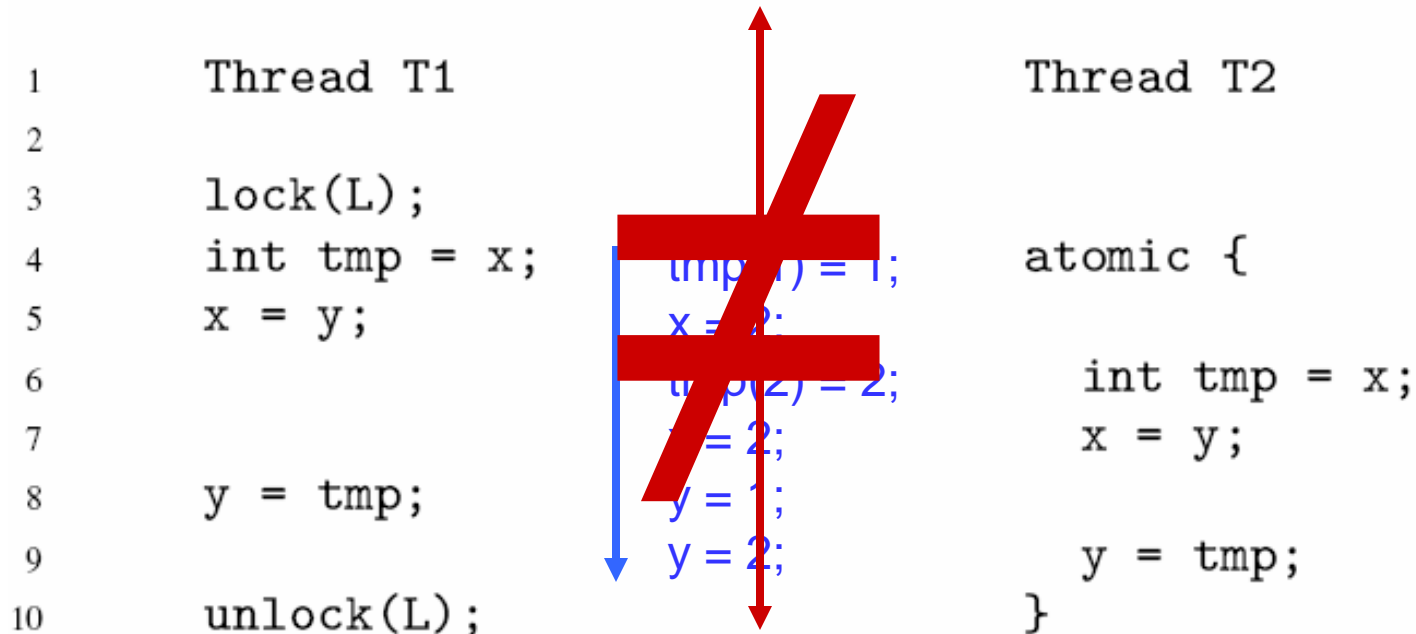
# Types of TM-Lock Conflicts

---

- **Locks Outside of Transactions (LoTs)**
  - Thread 1 runs lock
  - Thread 2 runs transaction
- **Locks Inside of Transactions (LiTs)**
  - Thread 1 runs transaction, calls locking code
  - Thread 2 runs transaction or lock

# LoT: Swap Violation

Given:  $x = 1, y = 2.$   
Swap:  $x = 2, y = 1.$



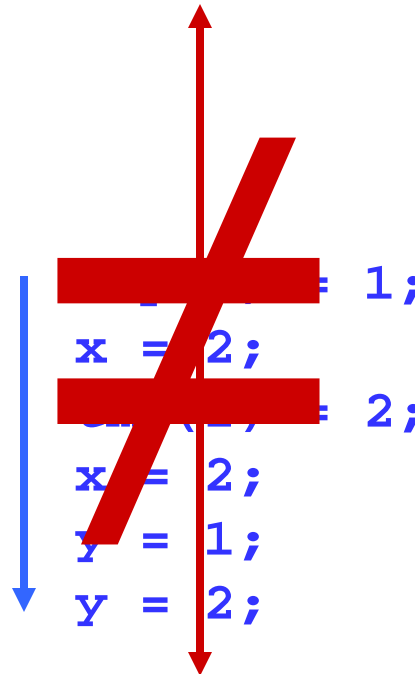
Result:  
 $x = 2, y = 2.$

# LiT: Swap Violation

Thread 1  
atomic {swapXY();}

```
void swapXY() {  
    lock(L);  
    int tmp = x;  
    x = y;  
  
    y = tmp;  
    unlock(L);  
}
```

Given:  $x = 1, y = 2.$   
Swap:  $x = 2, y = 1.$



Result:  
 $x = 2, y = 2.$

Thread 2  
atomic  
{

```
    int tmp = x;  
    x = y;  
  
    y = tmp;  
}
```

# Why Did They Fail?

---

- TM: optimistic concurrency
- Locks: pessimistic concurrency
- Transactions must respect mutual exclusion
- Full Lock Protection
  - LoT: TxLocks (Volos et al., TRANSACT '08)
  - LiT: atomic serialization (Ziarek et al., ECOOP '08)
  - If lock is acquired, all txes commit/abort
  - Requires *isolation*
- *Can we do better?*

# Our Novel LATM Policies

---

- **TM-Lock Protection**
  - Coder lists locks that conflict with txes
  - If conflict lock is acquired, all txes commit/abort
  - Uses *isolated txes for LiTs*
- **TX-Lock Protection**
  - Coder lists locks that conflict with a *specific* tx
  - If conflict lock is acquired, only *specific* tx C/A
  - Uses *irrevocable txes for LiTs*

# Throughput Comparison

---

- Full Lock Protection (existing)
  - LoTs: (txes) or (LoT + locks)
  - LiTs: (locks) or (txes) or (LiT)
- TM-Lock Protection (*new*)
  - LoTs: (non-conflict locks + txes) or (LoT + locks)
  - LiTs: (non-conflict locks + txes) or (LiT + locks)
- TX-Lock Protection (*new*)
  - LoTs: (non-conflict locks + txes) or (LoT + locks + txes)
  - LiTs: (non-conflict locks + txes) or (LiT + locks + txes)

# LoT Code: Full Lock Protection

```
// nothing todo
atomic {
    int tmp = x;
    x = y;
    y = tmp;
}
```

```
tx::lock(L);
int tmp = x;
x = y;
y = tmp;
tx::unlock(L);
```

# LoT Code: TM-Lock Protection

```
// before tx
lock_conflict(L);

atomic {
    int tmp = x;
    x = y;
    y = tmp;
}
```

```
tx::lock(L);
int tmp = x;
x = y;
y = tmp;
tx::unlock(L);
```

# LoT Code: TX-Lock Protection

```
atomic(t) {  
    // before tx work  
    t.lock_conflict(L);  
    int tmp = x;  
    x = y;  
    y = tmp;  
}
```

```
tx::lock(L);  
int tmp = x;  
x = y;  
y = tmp;  
tx::unlock(L);
```

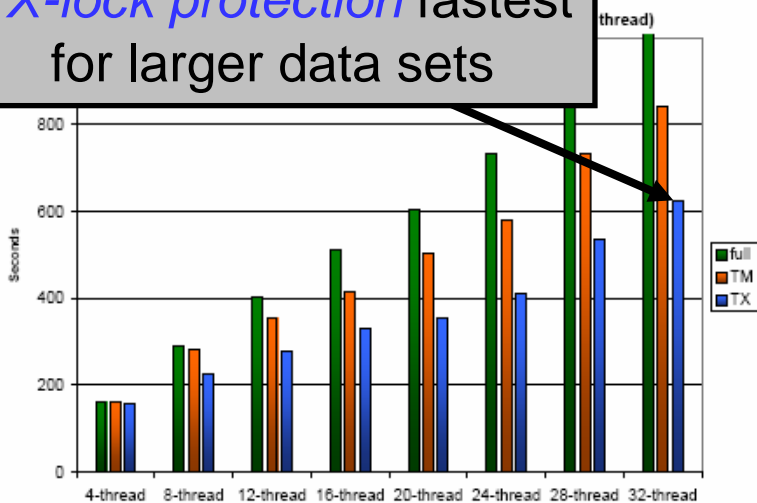
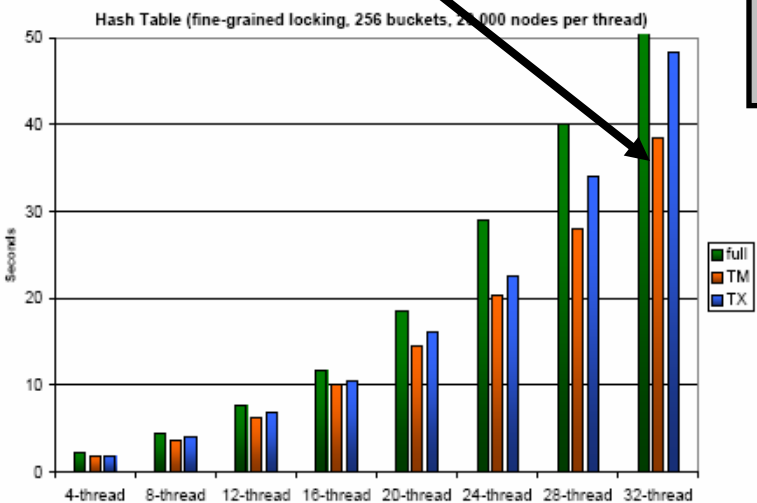
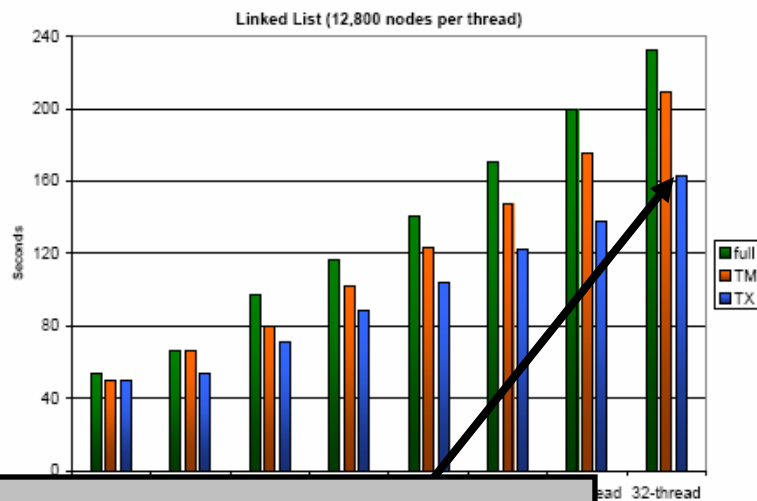
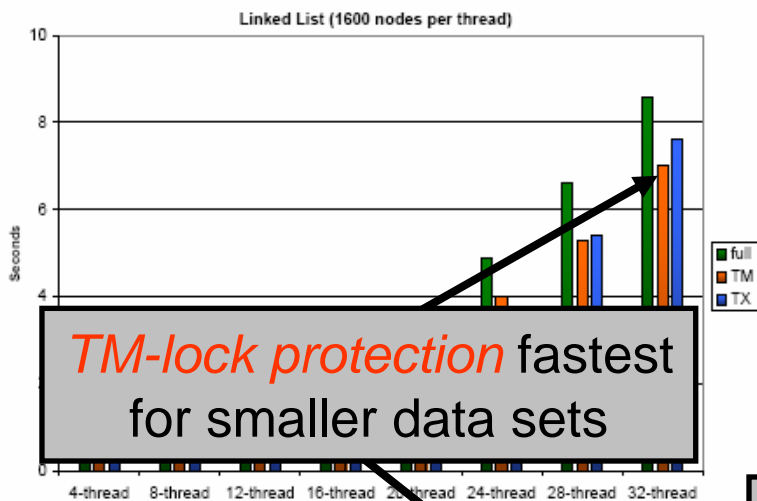
# LiT Code: TX-Lock Protection

```
atomic(t) {  
    // before tx work  
    t.lock_conflict(L);  
    swapXandY();  
}
```

Needed to prevent  
deadlocks and inform LoT  
txes (see paper for details)

```
void swapXandY()  
{  
    tx::lock(L);  
    int tmp = x;  
    x = y;  
    y = tmp;  
    tx::unlock(L);  
}
```

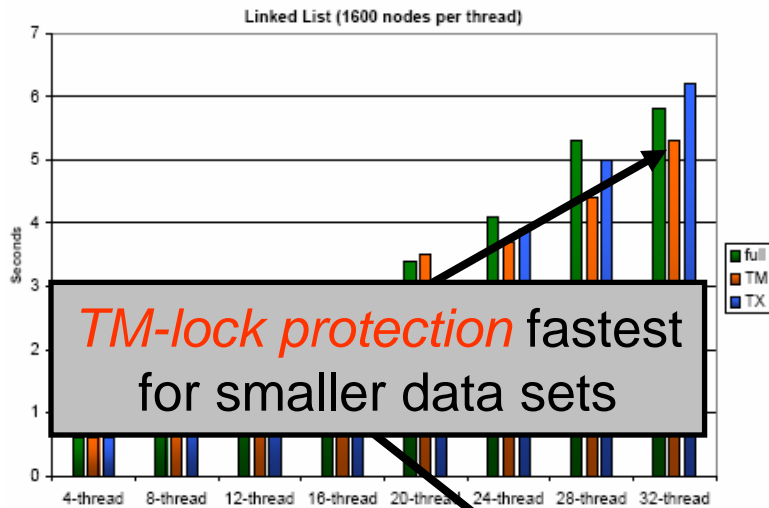
# LoTs: Experimental Results



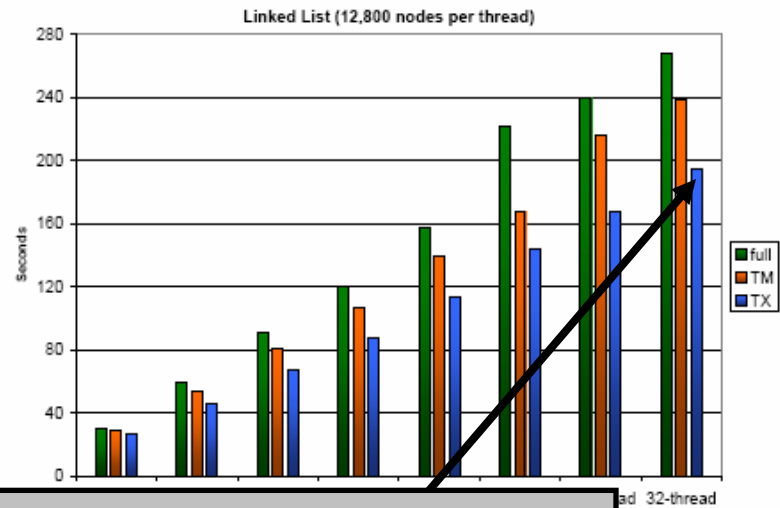
TM-lock protection fastest for smaller data sets

TX-lock protection fastest for larger data sets

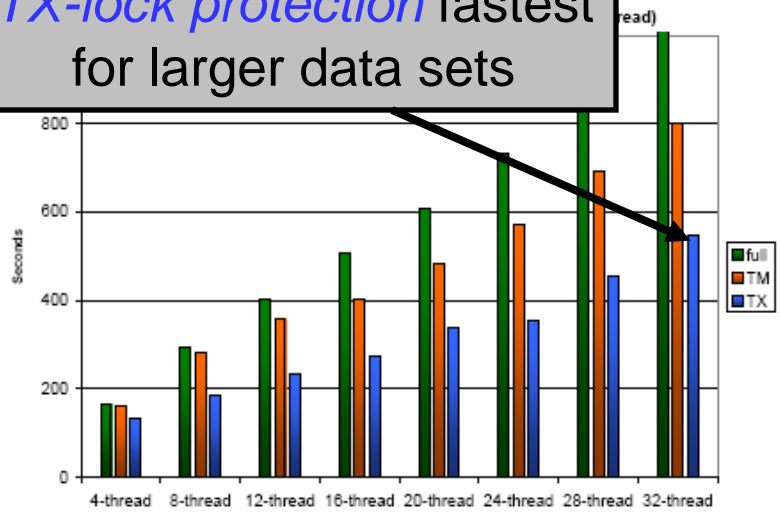
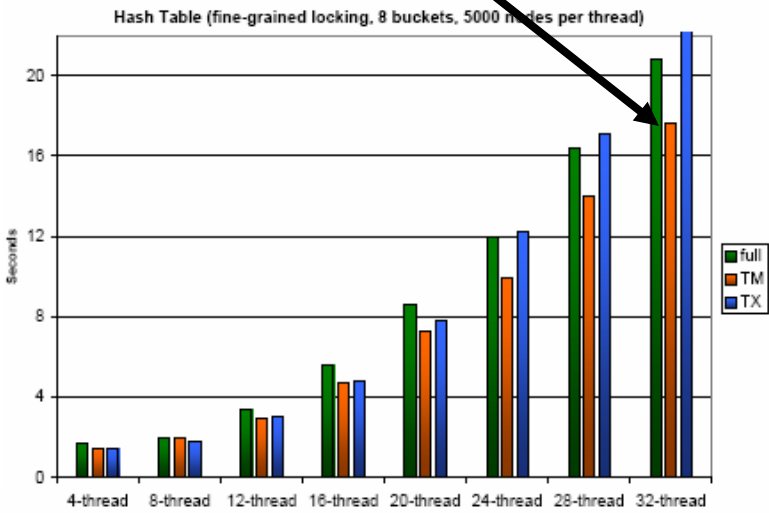
# LiTs: Experimental Results



*TM-lock protection* fastest for smaller data sets



*TX-lock protection* fastest for larger data sets



# Conclusion

---

- Locks + LATM = Correct Code
- Two new LATM policies
  - TM: some code, but faster than full
  - TX: more code, but fastest (for big data)
- Non-intuitive results
  - TM *outperforms* TX for small data

# Future Work

---

- **Combine static and dynamic LATM research (gradual system (?), Siek)**
- **Contention management**
  - *Past*: tx forward progress
  - *Future*: **txes + locks + etc.** forward progress
- **Formal proofs of LATM systems**
  - Lot of hand waving today (including our own!)

# Other Recent TM Publications

---

## **Boost + STM / Toward Simplified Parallel Support in C++**

[International Conference on Boost Libraries (BoostCon), May 2009]

## **Shifting the Parallel Programming Paradigm (Best Presentation Award)**

[Raytheon Information Systems and Computing (ISaC), March 2009]

## **Lock-Aware Transactional Memory**

[ACM International ASPLOS Conference, March 2009]

## **Optimizing Consistency Checking for Memory-Intensive Transactions**

[ACM International PODC Symposium, August 2008]

## **C++ Move Semantics for Exception Safety and Optimization in STM**

[International ICOOLPS Workshop, in conjunction with ECOOP, July 2008]

## **Extending Contention Managers for User-Defined Priority-Based Transactions**

[ACM EPHAM Workshop, in conjunction with CGO, April 2008]

## **DracoSTM: A Practical C++ Approach to Software Transactional Memory**

[ACM LCSD Symposium, in conjunction with OOPSLA, October 2007]

## **Exploration of Lock-Based Software Transactional Memory**

[MS Thesis, 2007]

# Comments / Questions?

Justin E. Gottschlich



University of Colorado at Boulder

Raytheon Company

[justin@nodeka.com](mailto:justin@nodeka.com)

