Hardware Transactional Memory

Daniel Schwartz-Narbonne

- Hardware Transactional Memories
- Hybrid Transactional Memories
- Case Study: Sun Rock
- Clever ways to use TM

Recap: Parallel Programming

1. Find independent tasks in the algorithm
2. Map tasks to execution units (e.g. threads)
3. Define and implement synchronization among tasks
    1. Avoid races and deadlocks, address memory model issues, …
4. Compose parallel tasks
5. Recover from errors
6. Ensure scalability
7. Manage locality
8. …

Transactional Memory

Recap: TM Implementation

Data Versioning

- Eager Versioning
- Lazy Versioning

Conflict Detection and Resolution

- Pessimistic Concurrency Control
- Optimistic Concurrency Control

Conflict Detection Granularity

- Object Granularity
- Word Granularity
- Cache line Granularity

Hardware vs. Software TM

Hardware Approach

- Low overhead
  - Buffers transactional state in Cache
- More concurrency
  - Cache-line granularity
- Bounded resource

Useful BUT Limited

Software Approach

- High overhead
  - Uses Object copying to keep transactional state
- Less Concurrency
  - Object granularity
- No resource limits

Useful BUT Limited

Kumar (Intel)

Hardware Transactional Memory

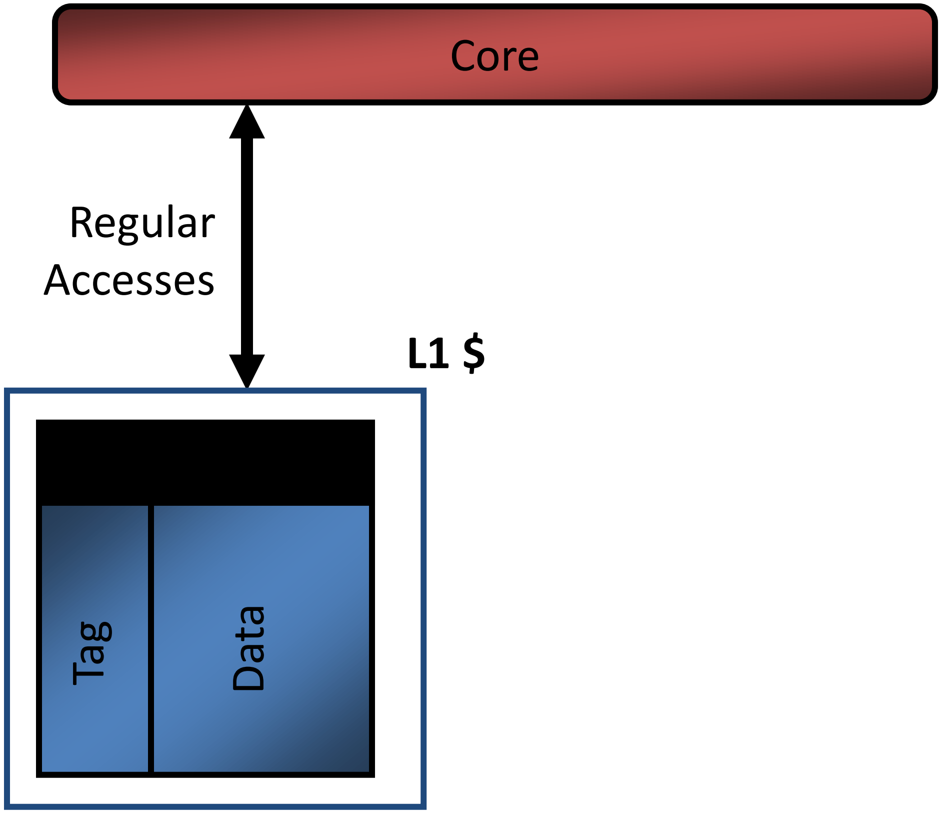
- Transactional memory implementations require tracking read / write sets
- Need to know whether other cores have accessed data we are using
- Expensive in software
  - Have to maintain logs / version ID in memory
  - Every read / write turns into several instructions
  - These instructions are inherently concurrent with the actual accesses, but STM does them in series

Hardware Transactional Memory

- Idea: Track read / write sets in Hardware
- Cache coherent hardware already manages much of this
- Basic idea: map storage to cache
- HTM is basically a smarter cache
  - Plus potentially some other storage buffers etc
- Can support many different TM paradigms
  - Eager, lazy
  - optimistic, pessimistic
- Default seems to be Lazy, pessimistic

HTM – The good

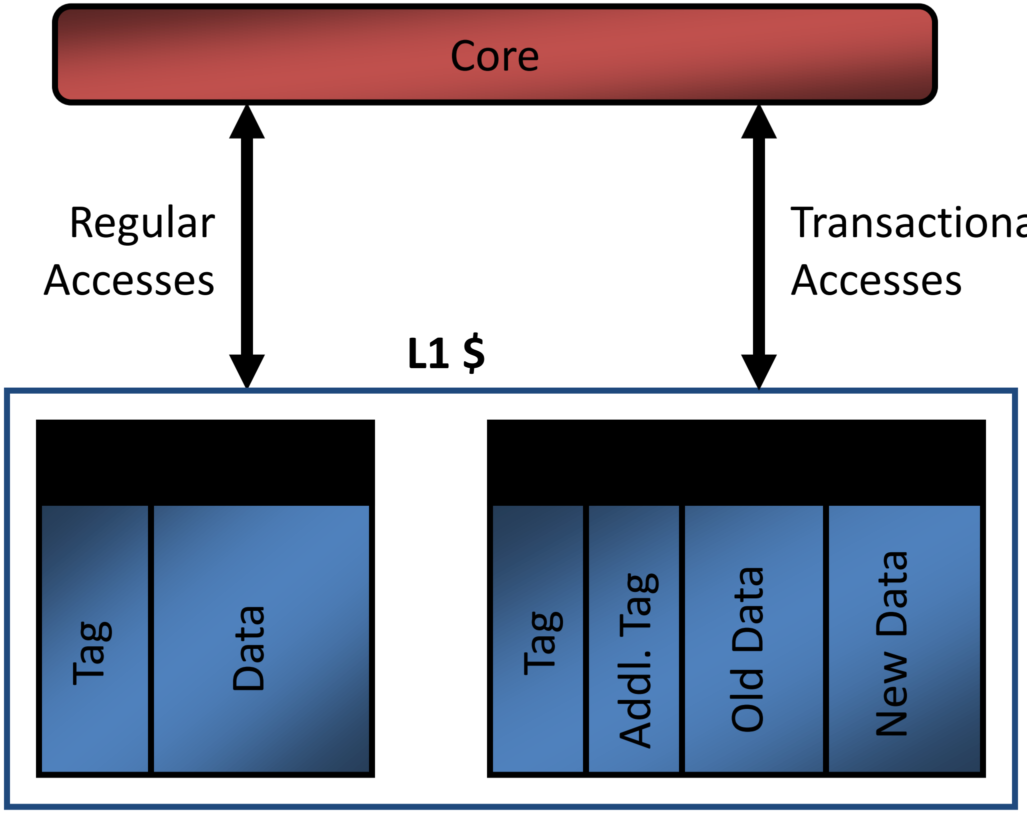
- Most hardware already exists
- Only small modification to cache needed



Kumar et al. (Intel)

HTM – The good

- Most hardware already exists
- Only small modification to cache needed



Kumar et al. (Intel)

HTM Example

Tag	data	Trans?	State	Tag	data	Trans?	state

Bus Messages:

```
atomic {
  read A
  write B =1
}
```

```
atomic {
  read B

  Write A = 2
}
```

HTM Example

Tag	data	Trans?	State	Tag	data	Trans?	state
				B	0	Y	S

Bus Messages: **2** read B

```
atomic {
  read A
  write B =1
}
```

```
atomic {
  read B


  Write A = 2
}
```

HTM Example

Tag	data	Trans?	State	Tag	data	Trans?	state
A	0	Y	S				
				B	0	Y	S

Bus Messages: **1** read A

```
atomic {
  read A
  write B =1
}
```

```
atomic {
  read B




  Write A = 2
}
```

HTM Example

Tag	data	Trans?	State	Tag	data	Trans?	state
A	0	Y	S				
B	1	Y	M	B	0	Y	S

Bus Messages: NONE

```
atomic {
  read A
  write B =1
}
```

```
atomic {
  read B



  Write A = 2
}
```

Conflict, visibility on commit

Tag	data	Trans?	State	Tag	data	Trans?	state
A	0	N	S				
B	1	N	M	B	0	Y	S

Bus Messages: **1** B modified

```
atomic {
  read A
  write B =1
}
```

```
atomic {
  read B


ABORT


  Write A = 2
}
```

Conflict, notify on write

Tag	data	Trans?	State	Tag	data	Trans?	state
A	0	Y	S				
B	1	Y	M	B	0	Y	S

Bus Messages: **1** speculative write to B
2: 1 conflicts with me

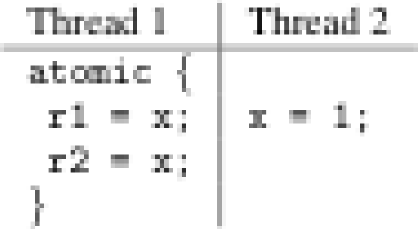
```
atomic {
  read A
  write B =1  ABORT?
}
```

```
atomic {
  read B


ABORT?


  Write A = 2
}
```

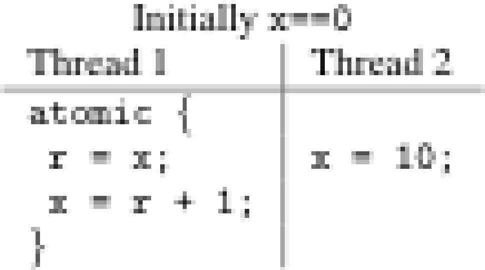
HTM – The good
Strong isolation



```
Thread 1      | Thread 2
--------------+---------
atomic {      |
  r1 = x;     | x = 1;
  r2 = x;     |
}             |
```

Can r1!=r2?

(a) Non-repeatable reads

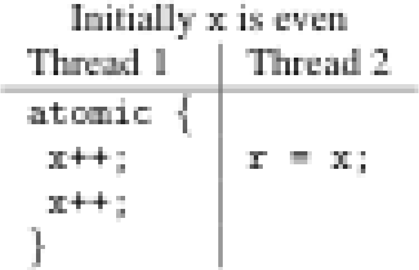


Initially x==0

```
Thread 1      | Thread 2
--------------+---------
atomic {      |
  r = x;      | x = 10;
  x = r + 1;  |
}             |
```

Can x==1?

(b) Lost updates



Initially x is even

```
Thread 1      | Thread 2
--------------+---------
atomic {      |
  x++;        | r = x;
  x++;        |
}             |
```

Can r be odd?

(c) Dirty reads

HTM – The good
ISA Extensions

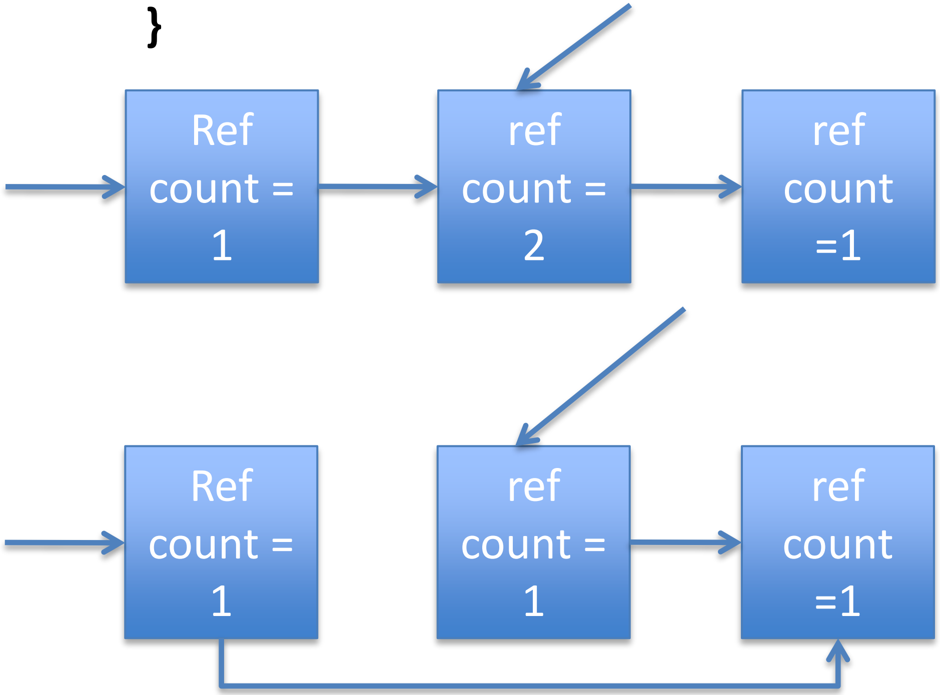
- Allows ISA extentions (new atomic operations)
- Double compare and swap
- Necessary for some non-blocking algorithms

```
int DCAS(int *addr1, int *addr2, int old1, int old2, int new1, int new2)
atomic {
        if ((*addr1 == old1) && (*addr2 == old2)) {
                *addr1 = new1;
                *addr2 = new2;
                return(TRUE);
        } else return(FALSE);
}
```

- Similar performance to handtuned java.util.concurrent implementation (Dice et al, ASPLOS ’09)

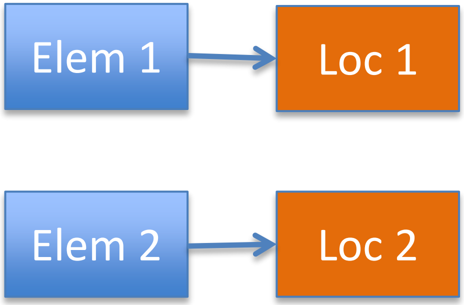
HTM – The good
ISA Extensions

```
int DCAS(int *addr1, int *addr2, int old1, int old2, int new1, int new2)
atomic {
        if ((*addr1 == old1) && (*addr2 == old2)) {
                *addr1 = new1;
                *addr2 = new2;
                return(TRUE);
        } else return(FALSE);
}
```



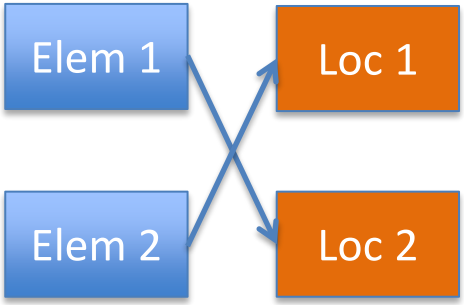
HTM – The good
ISA Extensions

- Allows ISA extentions (new atomic operations)
- Atomic pointer swap



HTM – The good
ISA Extensions

- Allows ISA extentions (new atomic operations)
- Atomic pointer swap



  - 21-25% speedup on canneal benchmark (Dice et al, SPAA’10)

HTM – The bad
False Sharing

Tag	data	Trans?	State	Tag	data	Trans?	state
				C/D	0/0	Y	S

Bus Messages: Read C/D

```
atomic {
  read A
  write D = 1
}
```

```
atomic {
  read C




  Write B = 2
}
```

HTM – The bad
False Sharing

Tag	data	Trans?	State	Tag	data	Trans?	state
				C/D	0/0	Y	S
A/B	0/0	Y	S				

Bus Messages: Read A/B

```
atomic {
  read A
  write D = 1
}
```

```
atomic {
  read C


  Write B = 2
}
```

HTM – The bad
False sharing

Tag	data	Trans?	State	Tag	data	Trans?	state
C/D	0/1	Y	M	C/D	0/0	Y	S
A/B	0/0	Y	S				

Bus Messages: Write C/D

```
atomic {
  read A
  write D = 1
}
```

```
atomic {
  read C

  Write B = 2
}
```

UH OH

HTM – The bad
Context switching

- Cache is unaware of context switching, paging, etc
- OS switching typically aborts transactions

HTM – The bad
Inflexible

- Poor support for advanced TM constructs
- Nested Transactions
- Open variables
- etc

HTM – The bad
Limited Size

Tag	data	Trans?	State	Tag	data	Trans?	state
A	0	Y	M				

Bus Messages: Read A

```
atomic {
  read A
  read B
  read C
  read D
}
  Write C/
```

HTM – The bad
Limited Size

Tag	data	Trans?	State	Tag	data	Trans?	state
A	0	Y	M				
B	0	Y	M				

Bus Messages: Read B

```
atomic {
  read A
  read B
  read C
  read D
}
```

HTM – The bad
Limited Size

Tag	data	Trans?	State	Tag	data	Trans?	state
A	0	Y	M				
B	0	Y	M				
C	0	Y	M				

Bus Messages: Read C

```
atomic {
  read A
  read B
  read C
  read D
}
```

HTM – The bad
Limited Size

Tag	data	Trans?	State	Tag	data	Trans?	state
A	0	Y	M				
B	0	Y	M				
C	0	Y	M				

Bus Messages: …

```
atomic {
  read A
  read B
  read C
  read D
}
```

UH OH

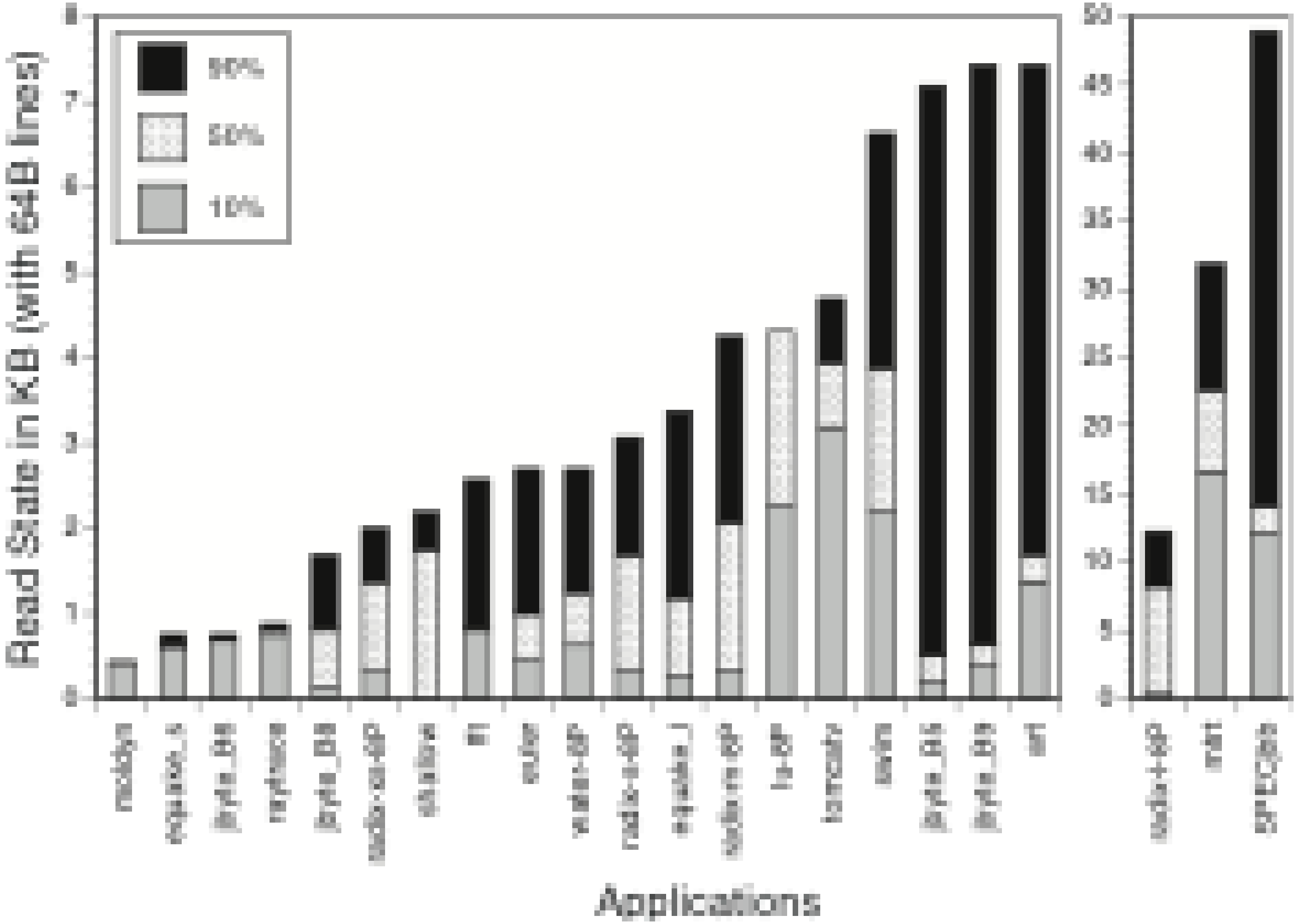


Figure 6: State read by individual transactions with store buffer granularity of 64-byte cache lines. We show state required by the smallest 10%, 50%, and 90% of iterations.

Transactional memory coherence and consistency (Hammond et al, ISCA ‘04)

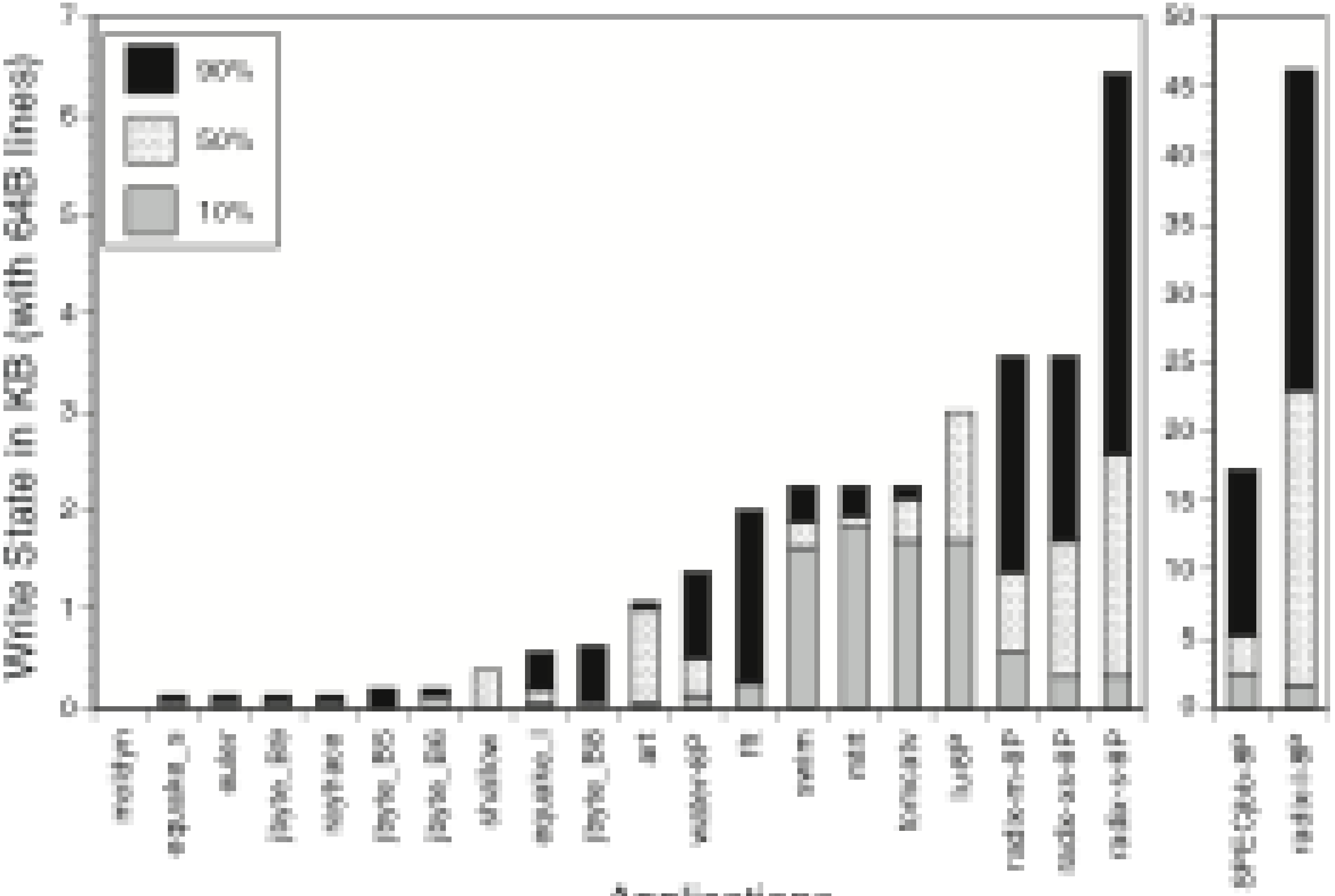


Figure 7: Same as Fig. 6, but for write state.

Transactional memory coherence and consistency (Hammond et al, ISCA ‘04)

Hardware vs. Software TM

Hardware Approach

- Low overhead
  - Buffers transactional state in Cache
- More concurrency
  - Cache-line granularity
- Bounded resource

Useful BUT Limited

Software Approach

- High overhead
  - Uses Object copying to keep transactional state
- Less Concurrency
  - Object granularity
- No resource limits

Useful BUT Limited

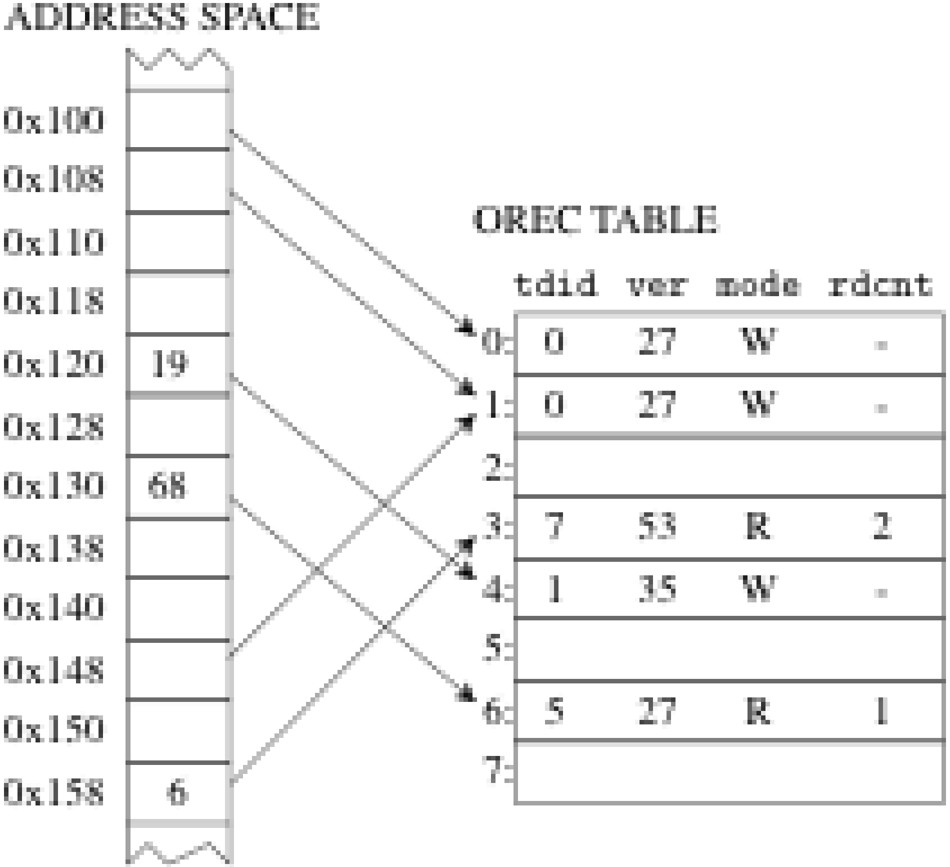
What if we could have both worlds simultaneously?

Kumar (Intel)

Hybrid TM

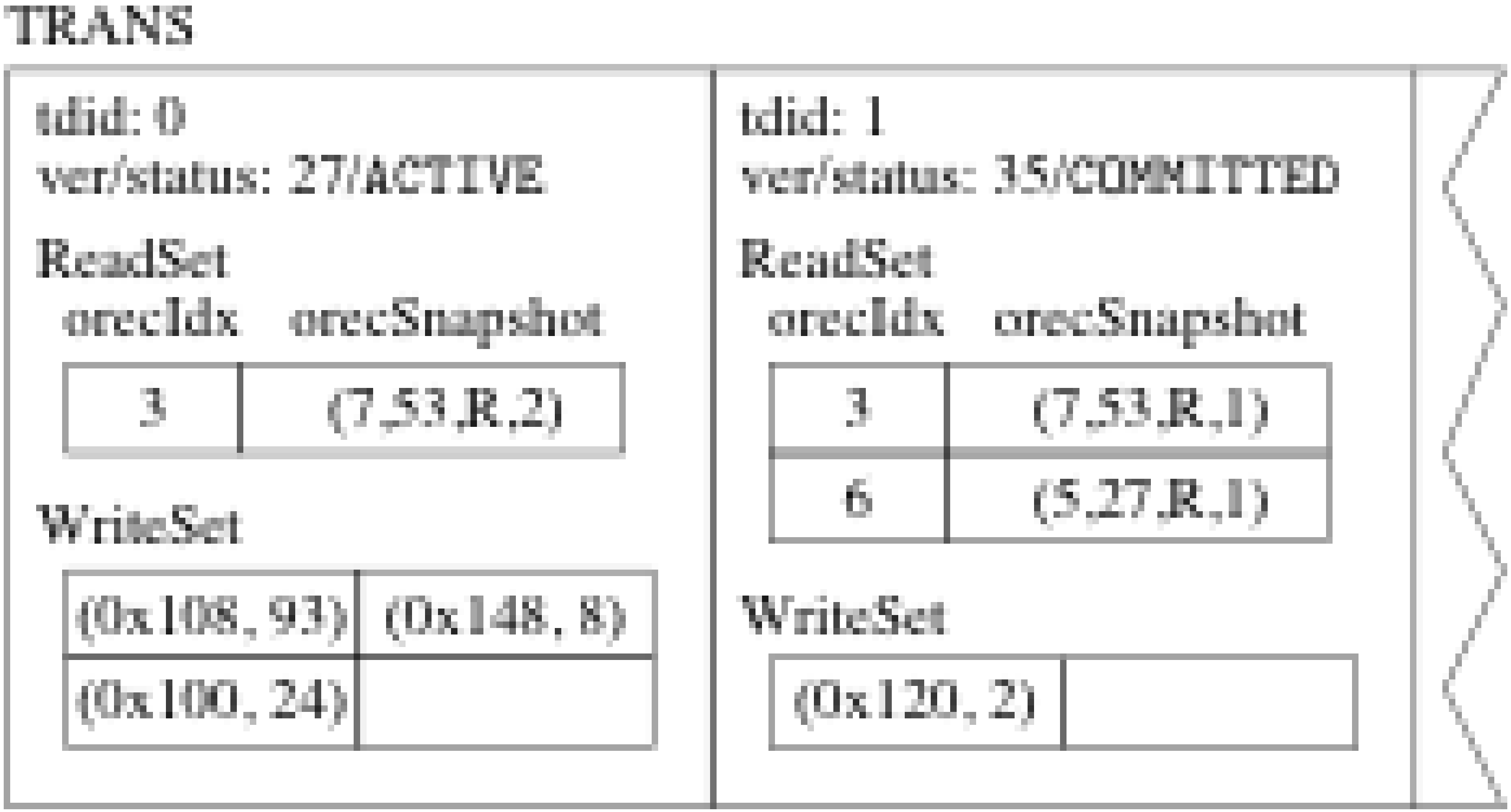
- Damron et al. ASPLOS ’06
- Pair software transactions with best-effort hardware transactions
- High level idea: software transactions maintain read/write state of variables, which hardware transactions can check
  - Similar to the cached bits in HTM
- Hashed table of per-object orecs
- Each record can be unowned, owned by 1 or more readers, or owned exclusive for writing

Hybrid TM



	tdid	ver	mode	rdcnt
0:	0	27	W	-
1:	0	27	W	-
2:				
3:	7	53	R	2
4:	1	35	W	-
5:				
6:	5	27	R	1
7:				

Hybrid TM



TRANS

tdid: 0
ver/status: 27/ACTIVE

ReadSet

orecIdx	orecSnapshot
3	(7,53,R,2)

WriteSet

(0x108, 93)	(0x148, 8)
(0x100, 24)	

tdid: 1
ver/status: 35/COMMITTED

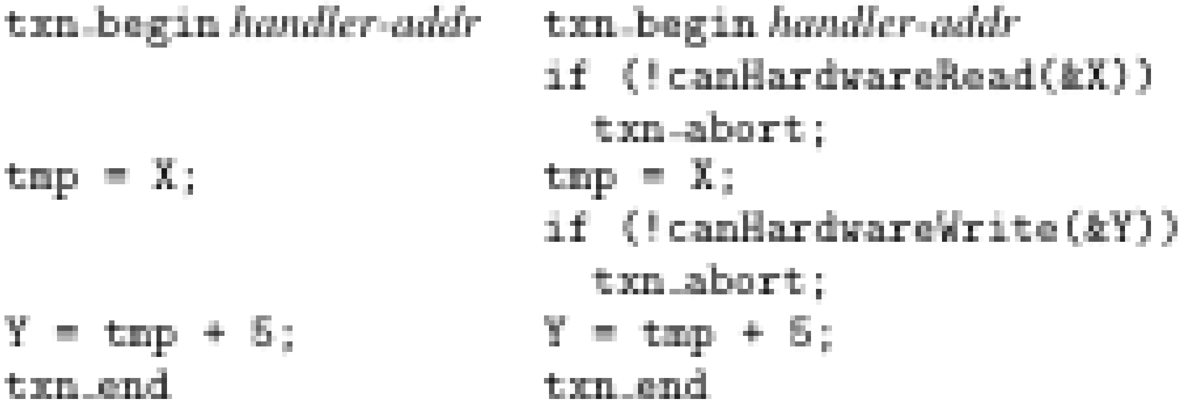
ReadSet

orecIdx	orecSnapshot
3	(7,53,R,1)
6	(5,27,R,1)

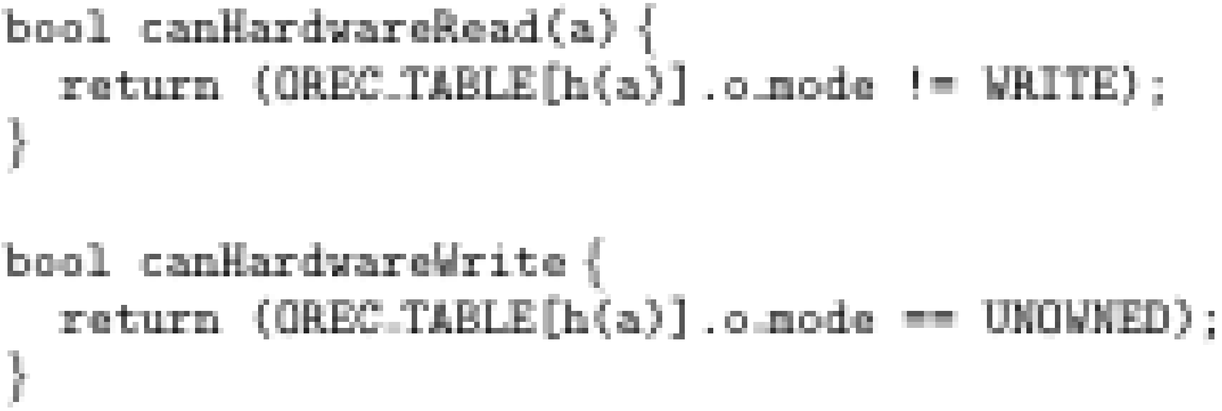
WriteSet

(0x120, 2)	

Hybrid TM



```
txn_begin handler-addr   txn_begin handler-addr
                         if (!canHardwareRead(&X))
                           txn_abort;
tmp = X;                 tmp = X;
                         if (!canHardwareWrite(&Y))
                           txn_abort;
Y = tmp + 5;             Y = tmp + 5;
txn_end                  txn_end
```



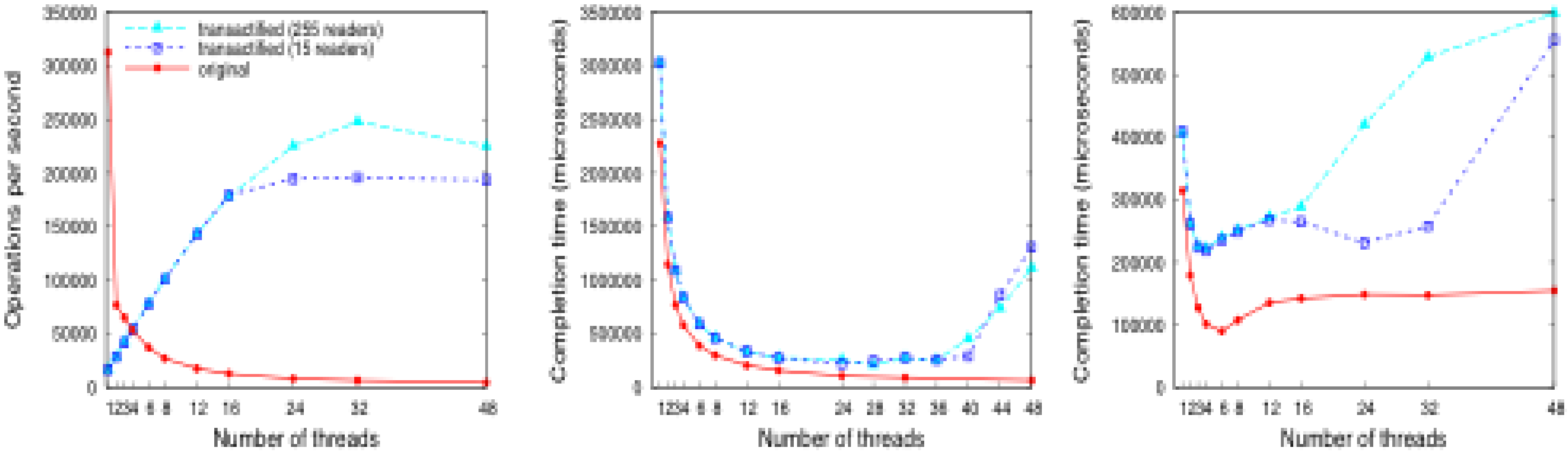
```
bool canHardwareRead(a) {
  return (OREC_TABLE[h(a)].o_mode != WRITE);
}

bool canHardwareWrite {
  return (OREC_TABLE[h(a)].o_mode == UNOWNED);
}
```

Hybrid TM

- Few problems:
- Change from unowned to read will spuriously fail transaction
- orec reading overhead unnecessary when only hardware transactions are running
  - Can maintain num_software_transactions variable, and avoid orec accesses when == 0

Hybrid TM



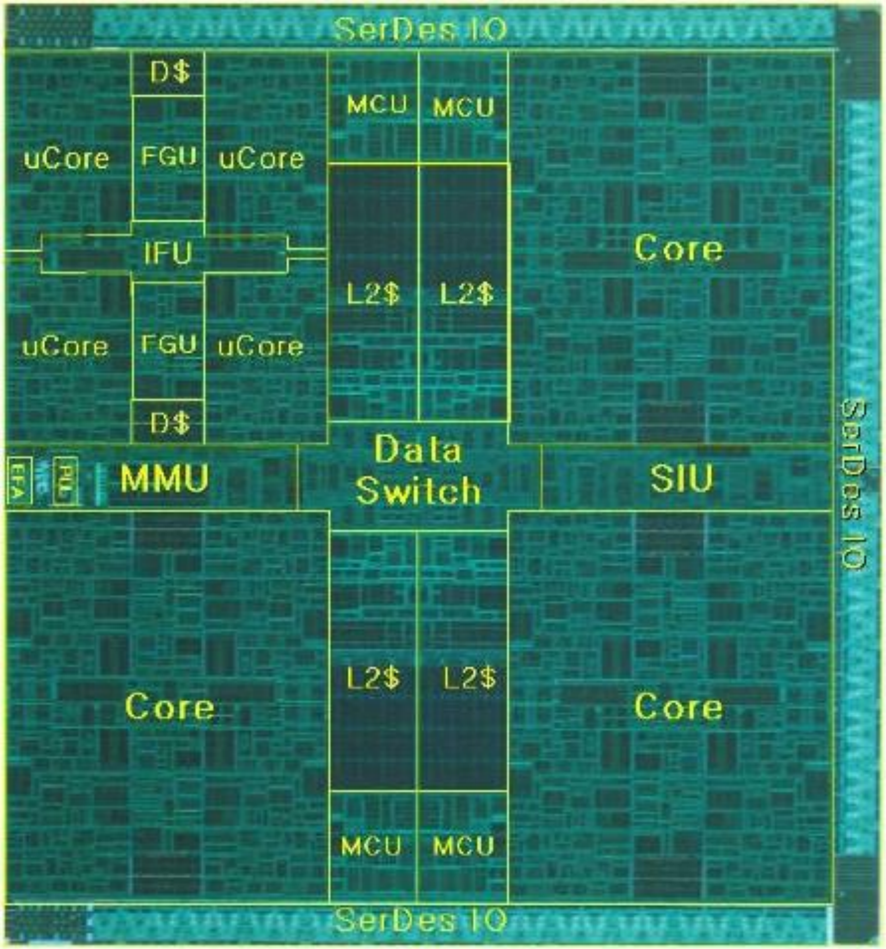
(a) (b) (c)

Figure 2. Software-only experiments: (a) Berkeley DB lock subsystem (b) `barnes` (c) `raytrace`

Case Study: SUN Rock

- Commercial processor with HTM support
- Sun actually built it, and was going to sell it
- Canceled by Oracle ☹
- Fascinating look into the real world challenges of HTM
- Dice, Lev, Moir and Nussbaum ASPLOS’09

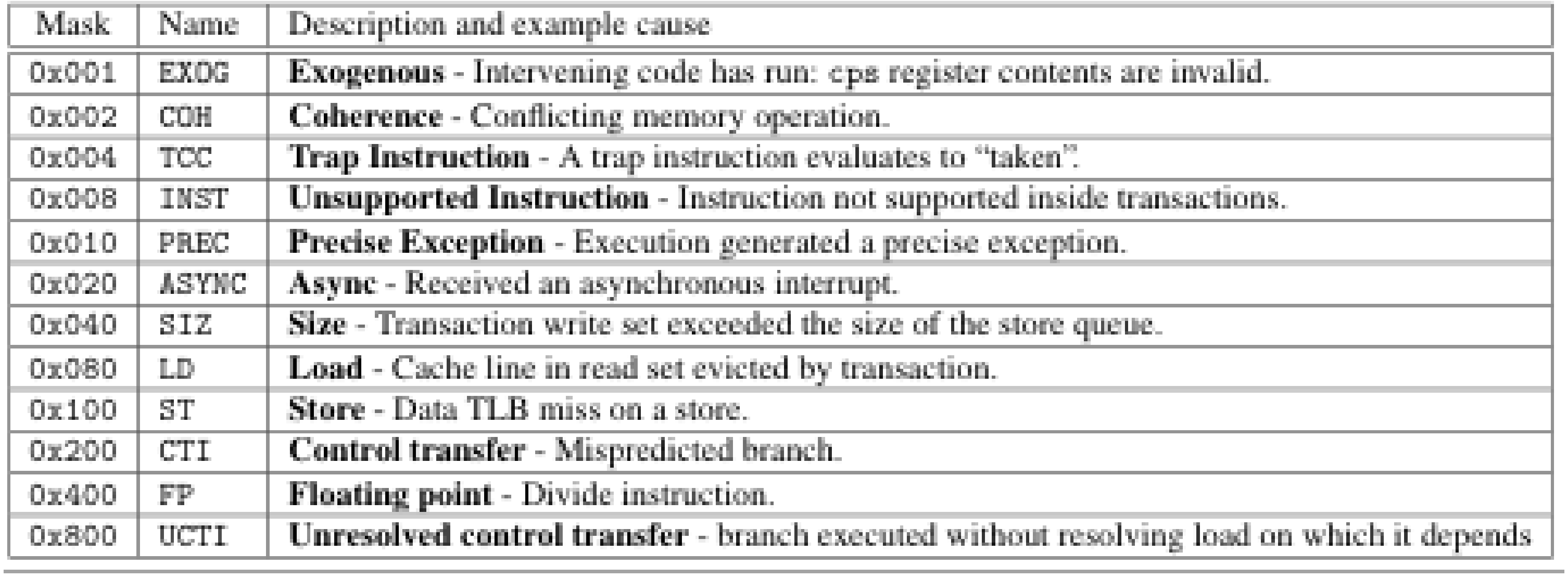
Case Study: Sun Rock



Case Study: SUN Rock

- Major challenge: Diagnosing the cause of Transaction aborts
  - Necessary for intelligent scheduling of transactions
  - Also for debugging code
  - And equally importantly, debugging the processor architecture / µarchitecture
- Many unexpected causes of aborts
- And Rock v1 diagnostics were unable to distinguish many distinct failure modes

Case Study: SUN Rock



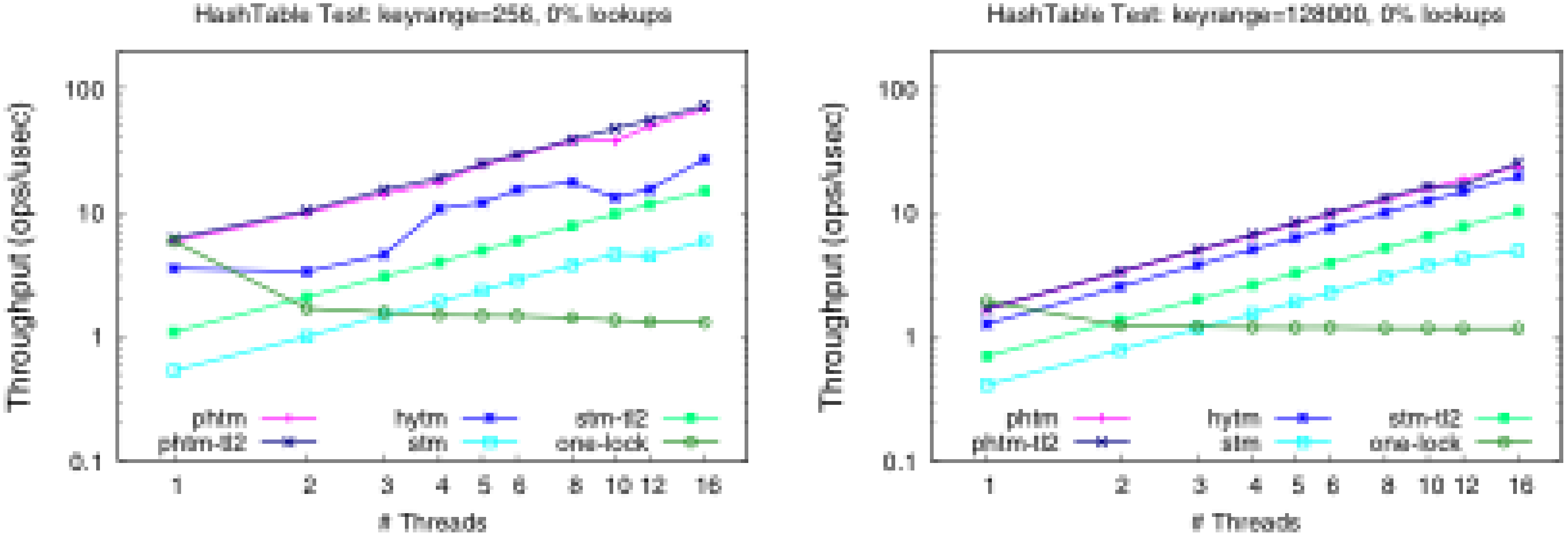
Mask	Name	Description and example cause
0x001	EXOG	**Exogenous** - Intervening code has run: cps register contents are invalid.
0x002	COH	**Coherence** - Conflicting memory operation.
0x004	TCC	**Trap Instruction** - A trap instruction evaluates to "taken".
0x008	INST	**Unsupported Instruction** - Instruction not supported inside transactions.
0x010	PREC	**Precise Exception** - Execution generated a precise exception.
0x020	ASYNC	**Async** - Received an asynchronous interrupt.
0x040	SIZ	**Size** - Transaction write set exceeded the size of the store queue.
0x080	LD	**Load** - Cache line in read set evicted by transaction.
0x100	ST	**Store** - Data TLB miss on a store.
0x200	CTI	**Control transfer** - Mispredicted branch.
0x400	FP	**Floating point** - Divide instruction.
0x800	UCTI	**Unresolved control transfer** - branch executed without resolving load on which it depends

Table 1. cps register: bit definitions and example failure reasons that set them.

Case Study: SUN Rock

- Several unexpected sources of aborts
- Branch mispredictions
  - Rock supports speculative branch execution. Mispredicted branches might invalidly cause the transaction to abort
- TLB misses
  - Context switches abort transactions. To get good performance, they found they had to warm the data structures using dummy CAS instructions
- Excessive cache misses
  - Rock hides cache miss latency using speculative buffers
  - If these buffers overflow, transaction must abort
- Core multithreading configuration
  - Each core can execute 2 threads in parallel, or one thread with twice the resources.

Case Study: SUN Rock



(a)

(b)

Figure 1. HashTable with 50% inserts, 50% deletes: (a) key range 256 (b) key range 128,000.

Case Study: SUN Rock

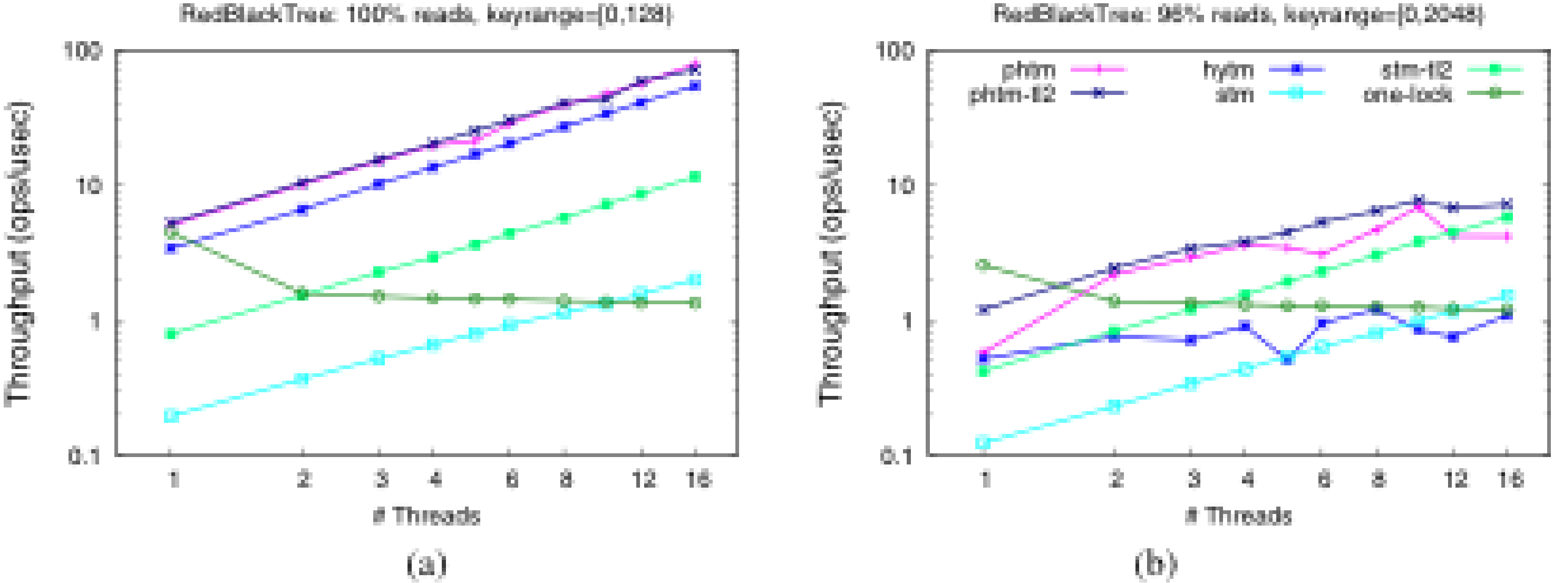
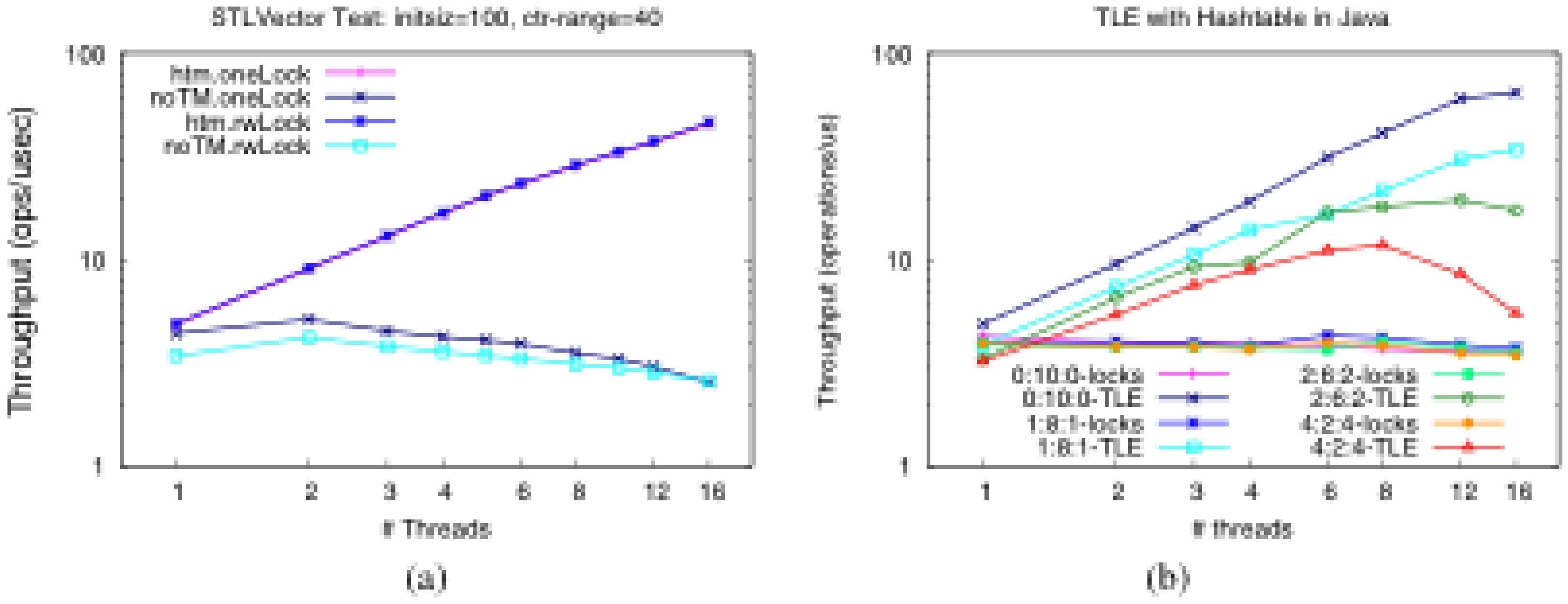


Figure 2. Red-Black Tree. (a) 128 keys, 100% reads (b) 2048 keys, 96% reads, 2% inserts, 2% deletes.

Case Study: SUN Rock



(a)

(b)

Figure 3. (a) TLE in C++ with STL `vector` (b) TLE in Java with `Hashtable`.

Clever Ways to use TM

- Lock Elision
  - In many data structures, accesses are contention free in the common case
  - But need locks for the uncommon case where contention does occur
  - For example, double ended queue
  - Can replace lock with atomic section, default to lock when needed
  - Allows extra parallelism in the average case

Lock Elision

```
hashTable.lock()
var = hashTable.lookup(X);
if (!var) hashTable.insert(X);
hashTable.unlock();
```

```
hashTable.lock()
var = hashTable.lookup(Y);
if (!var) hashTable.insert(Y);
hashTable.unlock();
```

Parallel Execution

```
atomic {
        if (!hashTable.isUnlocked()) abort;
        var = hashTable.lookup(X);
        if (!var) hashTable.insert(X);
} orElse …
```

```
atomic {
        if (!hashTable.isUnlocked()) abort;
        var = hashTable.lookup(X);
        if (!var) hashTable.insert(X);
} orElse …
```

Privatization

```
atomic {
        var = getWorkUnit();
        do_long_compution(var);
}
```

VS

```
atomic {
        var = getWorkUnit();
}
do_long_compution(var);
```

Note that this may only work correctly in STMs that support strong isolation

Work Deferral

```
atomic {
        do_lots_of_work();
        update_global_statistics();
}
```

Work Deferral

```
atomic {
        do_lots_of_work();
        update_global_statistics();
}
atomic {
        do_lots_of_work();
         atomic open {
                update_global_statistics();
        }
}
```

Work Deferral

```
atomic {
        do_lots_of_work();
        update_global_statistics();
}
atomic {
        do_lots_of_work();
         atomic open {
                update_global_statistics();
        }
}

atomic {
  do_lots_of_work();
  queue_up
  update_local_statistics(); //effectively serializes transactions
}
atomic{
        update_global_statististics_using_local_statistics()
}
```

} commit transaction(talk)

- Any questions?

Bibliography

- Chi Cao Minh, Martin Trautmann, JaeWoong Chung, Austen McDonald, Nathan Bronson, Jared Casper, Christos Kozyrakis, and Kunle Olukotun. 2007. **An effective hybrid transactional memory system with strong isolation guarantees**. *SIGARCH Comput. Archit. News* 35, 2 (June 2007), 69-80. DOI=10.1145/1273440.1250673 http://doi.acm.org/10.1145/1273440.1250673
- Bratin Saha, Ali-Reza Adl-Tabatabai, and Quinn Jacobson. 2006. **Architectural Support for Software Transactional Memory.** In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture* (MICRO 39). IEEE Computer Society, Washington, DC, USA, 185-196. DOI=10.1109/MICRO.2006.9 http://dx.doi.org/10.1109/MICRO.2006.9
- Dave Dice, Yossi Lev, Mark Moir, and Daniel Nussbaum. 2009. **Early experience with a commercial hardware transactional memory implementation**. In *Proceeding of the 14th international conference on Architectural support for programming languages and operating systems* (ASPLOS '09). ACM, New York, NY, USA, 157-168. DOI=10.1145/1508244.1508263 http://doi.acm.org/10.1145/1508244.1508263
- Dan Grossman. 2007. The transactional memory / garbage collection analogy. *SIGPLAN Not.* 42, 10 (October 2007), 695-706. DOI=10.1145/1297105.1297080 http://doi.acm.org/10.1145/1297105.1297080

Bibliography

- Milo Martin, Colin Blundell, and E. Lewis. 2006. **Subtleties of Transactional Memory Atomicity Semantics**. *IEEE Comput. Archit. Lett.* 5, 2 (July 2006), 17-. DOI=10.1109/L-CA.2006.18 http://dx.doi.org/10.1109/L-CA.2006.18
- Dave Dice , Ori Shalev , Nir Shavit **Transactional Locking II (2006)** In Proc. of the 20th Intl. Symp. on Distributed Computing
- Lance Hammond, Vicky Wong, Mike Chen, Brian D. Carlstrom, John D. Davis, Ben Hertzberg, Manohar K. Prabhu, Honggo Wijaya, Christos Kozyrakis, and Kunle Olukotun. 2004. **Transactional Memory Coherence and Consistency**. In *Proceedings of the 31st annual international symposium on Computer architecture* (ISCA '04). IEEE Computer Society, Washington, DC, USA, 102-.
- ***Rock*: A SPARC CMT Processor** *www.**hotchips**.org/archives/hc20/3_Tues/HC20.26.931.pdf*
- Peter Damron, Alexandra Fedorova, Yossi Lev, Victor Luchangco, Mark Moir, and Daniel Nussbaum. 2006. **Hybrid transactional memory**. *SIGOPS Oper. Syst. Rev.* 40, 5 (October 2006), 336-346. DOI=10.1145/1168917.1168900 http://doi.acm.org/10.1145/1168917.1168900
- Tatiana Shpeisman, Vijay Menon, Ali-Reza Adl-Tabatabai, Steven Balensiefer, Dan Grossman, Richard L. Hudson, Katherine F. Moore, and Bratin Saha. 2007. **Enforcing isolation and ordering in STM**. *SIGPLAN Not.* 42, 6 (June 2007), 78-88. DOI=10.1145/1273442.1250744 http://doi.acm.org/10.1145/1273442.1250744