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

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

Useful BUT Limited
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

<table>
<thead>
<tr>
<th>Tag</th>
<th>data</th>
<th>Trans?</th>
<th>State</th>
<th>Tag</th>
<th>data</th>
<th>Trans?</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bus Messages:**

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

```plaintext
atomic {
  read B
  Write A = 2
}
```
## HTM Example

<table>
<thead>
<tr>
<th>Tag</th>
<th>data</th>
<th>Trans?</th>
<th>State</th>
<th>Tag</th>
<th>data</th>
<th>Trans?</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>B</td>
<td>O</td>
<td>Y</td>
<td>S</td>
</tr>
</tbody>
</table>

Bus Messages: 2 read B

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

atomic {
    read B
    write A = 2
}
```
## HTM Example

<table>
<thead>
<tr>
<th>Tag</th>
<th>data</th>
<th>Trans?</th>
<th>State</th>
<th>Tag</th>
<th>data</th>
<th>Trans?</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>Y</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>Y</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bus Messages: 1 read A

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

atomic {
  read B
}

Write A = 2
```plaintext
}
```
## HTM Example

### Table:

<table>
<thead>
<tr>
<th>Tag</th>
<th>data</th>
<th>Trans?</th>
<th>State</th>
<th>Tag</th>
<th>data</th>
<th>Trans?</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>Y</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>Y</td>
<td>M</td>
<td>B</td>
<td>0</td>
<td>Y</td>
<td>S</td>
</tr>
</tbody>
</table>

### Bus Messages:

- NONE

### Code:

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

atomic {
    read B
    Write A = 2
}
```
Conflict, visibility on commit

<table>
<thead>
<tr>
<th>Tag</th>
<th>data</th>
<th>Trans?</th>
<th>State</th>
<th>Tag</th>
<th>data</th>
<th>Trans?</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>N</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>N</td>
<td>M</td>
<td>B</td>
<td>0</td>
<td>Y</td>
<td>S</td>
</tr>
</tbody>
</table>

Bus Messages: 1 B modified

atomic {
    read A
    write B =1
}

atomic {
    read B
    ABORT
    Write A = 2
}
## Conflict, notify on write

<table>
<thead>
<tr>
<th>Tag</th>
<th>data</th>
<th>Trans?</th>
<th>State</th>
<th>Tag</th>
<th>data</th>
<th>Trans?</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>Y</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>Y</td>
<td>M</td>
<td>B</td>
<td>0</td>
<td>Y</td>
<td>S</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>atomic</code></td>
<td></td>
</tr>
<tr>
<td><code>r1 = x;</code></td>
<td><code>x = 1;</code></td>
</tr>
<tr>
<td><code>r2 = x;</code></td>
<td></td>
</tr>
</tbody>
</table>

Can `r1 != r2`?
(a) Non-repeatable reads

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>atomic</code></td>
<td></td>
</tr>
<tr>
<td><code>r = x;</code></td>
<td></td>
</tr>
<tr>
<td><code>x = r + 1;</code></td>
<td></td>
</tr>
</tbody>
</table>

Can `x == 1`?
(b) Lost updates

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>atomic</code></td>
<td></td>
</tr>
<tr>
<td><code>x++;</code></td>
<td><code>r = x;</code></td>
</tr>
<tr>
<td><code>x++;</code></td>
<td></td>
</tr>
</tbody>
</table>

Can `r` be odd?
(c) Dirty reads
HTM – The good ISA Extensions

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

```c
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

```c
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 extensions (new atomic operations)
• Atomic pointer swap

Elem 1 → Loc 1
Elem 2 → Loc 2
HTM – The good ISA Extensions

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

- 21-25% speedup on canneal benchmark (Dice et al, SPAA’10)
HTM – The bad False Sharing

<table>
<thead>
<tr>
<th>Tag</th>
<th>data</th>
<th>Trans?</th>
<th>State</th>
<th>Tag</th>
<th>data</th>
<th>Trans?</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C/D</td>
<td>0/0</td>
<td>Y</td>
<td>S</td>
</tr>
</tbody>
</table>

Bus Messages: Read C/D

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

```plaintext
atomic {
  read C
  Write B = 2
}
```
## HTM – The bad False Sharing

<table>
<thead>
<tr>
<th>Tag</th>
<th>data</th>
<th>Trans?</th>
<th>State</th>
<th>Tag</th>
<th>data</th>
<th>Trans?</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C/D</td>
<td>0/0</td>
<td>Y</td>
<td>S</td>
</tr>
<tr>
<td>A/B</td>
<td>0/0</td>
<td>Y</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bus Messages: Read A/B

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

atomic {
  read C
  write B = 2
}
HTM – The bad False sharing

<table>
<thead>
<tr>
<th>Tag</th>
<th>data</th>
<th>Trans?</th>
<th>State</th>
<th>Tag</th>
<th>data</th>
<th>Trans?</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/D</td>
<td>0/1</td>
<td>Y</td>
<td>M</td>
<td>C/D</td>
<td>0/0</td>
<td>Y</td>
<td>S</td>
</tr>
<tr>
<td>A/B</td>
<td>0/0</td>
<td>Y</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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
The bad
Limited Size

Bus Messages: Read A

atomic {
read A
read B
read C
read D
}
Write C/
## HTM – The bad Limited Size

<table>
<thead>
<tr>
<th>Tag</th>
<th>data</th>
<th>Trans?</th>
<th>State</th>
<th>Tag</th>
<th>data</th>
<th>Trans?</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>Y</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>Y</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Bus Messages: Read B

```lang-none
atomic {
    read A
    read B
    read C
    read D
}
```
## HTM – The bad Limited Size

<table>
<thead>
<tr>
<th>Tag</th>
<th>data</th>
<th>Trans?</th>
<th>State</th>
<th>Tag</th>
<th>data</th>
<th>Trans?</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>Y</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>Y</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>Y</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bus Messages: Read C

```plaintext
atomic {
    read A
    read B
    read C
    read D
}
```
**HTM – The bad Limited Size**

<table>
<thead>
<tr>
<th>Tag</th>
<th>data</th>
<th>Trans?</th>
<th>State</th>
<th>Tag</th>
<th>data</th>
<th>Trans?</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>Y</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>Y</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>Y</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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

**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?
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 oreces
• Each record can be unowned, owned by 1 or more readers, or owned exclusive for writing
Hybrid TM
### Hybrid TM

**TRANS**

<table>
<thead>
<tr>
<th>tdid: 0</th>
<th>tdid: 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ver/status: 27/ACTIVE</td>
<td>ver/status: 35/COMMITTED</td>
</tr>
</tbody>
</table>

#### ReadSet

<table>
<thead>
<tr>
<th>oreclIdx</th>
<th>orecSnapshot</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>(7,53,R,2)</td>
</tr>
</tbody>
</table>

#### WriteSet

<table>
<thead>
<tr>
<th>oreclIdx</th>
<th>orecSnapshot</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0x108, 93)</td>
<td>(0x148, 8)</td>
</tr>
<tr>
<td>(0x100, 24)</td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>oreclIdx</th>
<th>orecSnapshot</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>(7,53,R,1)</td>
</tr>
</tbody>
</table>

#### WriteSet

<table>
<thead>
<tr>
<th>oreclIdx</th>
<th>orecSnapshot</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>(5,27,R,1)</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>oreclIdx</th>
<th>orecSnapshot</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0x120, 2)</td>
<td></td>
</tr>
</tbody>
</table>
Hybrid TM

txn.begin handler-addr
    if (!canHardwareRead(&X))
        txn.abort;
    tmp = X;
    if (!canHardwareWrite(&Y))
        txn.abort;
    Y = tmp + 5;
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

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

• 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

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

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

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

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_computation(var);
}

VS

atomic {
    var = getWorkUnit();
}
    do_long_computation(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

```c
atomic {
    do_lots_of_work();
    update_global_statistics();
}
atomic {
    do_lots_of_work();
    atomic open { atomic open {
        update_global_statistics();
    }
    }
}
```
Work Deferral

```plaintext
atomict {
    do_lots_of_work();
    update_global_statistics();
}

atomict {
    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_statistics_using_local_statistics()
}
```
commit transaction(talk)

• Any questions?


Bibliography

- Dave Dice, Ori Shalev, Nir Shavit Transactional Locking II (2006) In Proc. of the 20th Intl. Symp. on Distributed Computing