Transactional Memory in Java™ Technology based Systems

Presenters:
Suresh Srinivas, Principal Engineer,
Vyacheslav Shakin, SW Engineering Manager,
Software & Solutions Group, Intel
TS-6316

Contributors:
Vijay Menon, Naveen Neelakantam, Adam Welc, Bratin Saha, Ali-Reza Adl-Tabatabai, Yang Ni, Tatiana Shpeisman, Rick Hudson,
Yuri Dolgov, Boris Kuznetsov, Evgueni Brevnov, Dmitry Yershov, Mingqiu Sun

* Other brands and names are the property of their respective owners
Learn about Transactional Memory and how Java™ Technology-based Systems can utilize both Software and Hardware Transactional Memory.
Agenda

> Introduction
  • Concurrency
  • Transactional Memory
> Software Transactional Memory
> Hardware Transactional Memory
> Results
> Summary & Call to Action
### Multicore Era Upon Us – Intel Perspective

<table>
<thead>
<tr>
<th>Multi-core Acceleration</th>
<th>Dawn of multi-core era</th>
</tr>
</thead>
</table>
January 2006: Intel ® Core™ Duo  
July 2006: Intel ® Core™ 2 Duo  
November 2006: Core 2 Extreme QX6700 |

http://www.intel.com/multi-core/
Concurrency Control Vectors

> Granularity
  - Coarse Grain
  - Fine Grain

> Scalability
  - Scale Up
  - Scale Out

> Partitioning
  - Task Level Parallelism
  - Data Parallelism
  - Main Core/Accelerator
Concurrency Vectors - Granularity

> Coarse Grain Concurrency
  - Business Level Java 2 Platform, Enterprise Edition (J2EE) Transaction (every connection is concurrently processed)
    - Commercial Application Servers
    - Open Source Application Servers
  - Application Level Concurrency
    - Virtualization
    - Multiple Application Processes or Java Virtual Machine (JVM™) instances

> Fine Grain Concurrency
  - Data Structure Level
    - java.util.concurrent Data Structures
  - Transactional Memory
  - Loop Level
  - Thread Level
TM Introduction

> Transactional Memory: Sequence of memory operations that execute completely (commit) or have no effect (abort)

> Software Transactional Memory
  • Software Only Implementation
  • New Language Constructs

> Hardware Transactional Memory
  • Hardware Only or HW/SW Implementation
  • New Instructions
Software Transactional Memory

- Implemented completely in software
- Language Extensions (atomic, retry)
- Software System guarantees atomicity, isolation
- Software System discovers concurrency (multiple readers, concurrent writers to disjoint data)
- Coarse Grain Programming and Fine Grain Performance

```java
public Object get (Object k) {
   synchronized (MapLock) {
      return Map.get(k);
   }
}

public Object get (Object k) {
   atomic {
      return Map.get(k);
   }
}
```
Hardware Transactional Memory

- Hardware views individual threads as executing reads/writes to shared memory
- In a transaction we want the series of reads/writes to occur at a single instant in time
  - Intermediate States are not visible
  - The transaction is “committed” when the state is made visible
  - If the transaction “aborts” due to “conflicts” the changes must be rolled back
- Idea: Utilize Caches/Coherence in Modern Microprocessors
- Multiple Implementations Possible

Transactional Memory is a tool for Concurrency Control In Multi Core Systems
Agenda

- Introduction to Concurrency Transactional Memory
- Software Transactional Memory
  - Examples
  - Design
  - Implementation
- Hardware Transactional Memory
- Results
- Summary & Call to Action
What is wrong with this code?

Old JDK™ release Example from Cormac Flanagan

```java
class StringBuffer {
    public synchronized void append(StringBuffer sb) {
        int len = sb.length();
        if (this.count + len > this.value.length) {
            this.expand(...);
            sb.getChars(0, len, this.value, this.count);
        }
    }
}
```

http://www.soe.ucsc.edu/~cormac/
Software Transactional Memory Example

Another thread could have modified `sb` & the use of `len` later on might not be valid. Declaring `atomic` to protect against `sb` updates fixes this.

```java
class StringBuffer {
    public void append(StringBuffer sb) {
        atomic{
            int len = sb.length();
            if(this.count + len > this.value.length)
                this.expand(...);
            sb.getChars(0,len,this.value,this.count);
        }
    }
}
```

Another thread could have modified `sb` & the use of `len` later on might not be valid. Declaring `atomic` to protect against `sb` updates fixes this.
Queue Example: Lock Challenges

```java
static void move(Queue q1, Queue q2) {
    synchronized (q1) {
        synchronized (q2) {
            v = q1.dequeue();
            q2.enqueue(v);
        }
    }
}
```

`qa = new Queue(...);`

`qb = new Queue(...);`

**T1:**
Queue.move(qa, qb);

**T2:**
Queue.move(qb, qa);

**T1:**
Acquire qa lock

**T2:**
Acquire qb lock

Wait for qa lock

Wait for qb lock

Deadlock since locks don’t compose
Queue Example: STM Solution

```java
static void move(Queue q1, Queue q2) {
    atomic {
        v = q1.dequeue();
        q2.enqueue(v);
    }
}
```

```java
qa = new Queue(...);
qb = new Queue(...);

T1:
Queue.move(qa, qb);
T2:
Queue.move(qb, qa);
```

Atomic commit one at a time

T1:
Atomic_start
...  
Atomic_commit

T2:
Atomic_start
...
Atomic_abort
Atomic_start  // retry
...
Atomic_commit
STM Example Summary

```java
static void move(Queue q1, Queue q2) {
    synchronized (q1) {
        synchronized (q2) {
            v = q1.dequeue();
            q2.enqueue(v);
        }
    }
    static void move(Queue q1, Queue q2) {
        atomic {
            v = q1.dequeue();
            q2.enqueue(v);
        }
    }

    Focus is on “what” rather than the “how”
```
Software Transactional Memory Design

> Isolation
  - Weak/Strong Atomic System

> Data Management
  - In-Place/Buffered

> Conflict Detection
  - Eager, Lazy, Granularity, Policies

> Language Integration
  - Library/Language Extension/New Language
  - Native Code, I/O, Exceptions
  - Nesting
Isolation

- Each transaction must appear to execute in complete isolation relative to ....
- Weak Atomic Systems – Only relative to other transactions
- Strong Atomic Systems – Relative to both transactional and non-transactional access
- Weak is not a strict Subset of Strong
- Most Software Transactional Memory systems provide Weak Atomicity Guarantees
Language Integration – Implementations

Most Easy

Integration with Existing Java programming language Semantics

Least Easy

DSTM2*
- Library based approach
- Atomicity in “Data”
- VM Agnostic
- Flexible Framework

AtomJava*
- Language based approach
- Strong Atomicity
- VM Agnostic

McRT STM
- Language based approach
- Strong and Weak Atomicity
- Flexible Framework
- VM/JIT dependent for optimizations

Atomos*
- Language based approach
- Strong Atomicity
- Changes Java to not have wait/notify/notifyAll

X10*
- New HPCS languages
- Language provides varied TM support

Our approach

Most Easy
Integration with Existing Java programming language Semantics
Least Easy

* Other brands and names are the property of their respective owners
Java Technology-based STM System

- Pure software implementation
- Language extensions in Java programming language
- Deep integration into virtual machine & dynamic compiler
  - RISC like STM API
  - Compiler optimizations
  - Varying granularity for conflict detection
  - Complete garbage collection support
- Weak/Strong Atomic System
- No I/O in Transactions
- Limited Native Code in Transactions
Java Technology-based STM System Overview

Transformer

Transactional Java

Java + STM API

Transactional IR

Optimized T-IR

Native Code

Enhanced JIT

Enhanced VM

McRT STM
Transactional Java Platform

Java platform + new language constructs:

- Atomic: execute block atomically
  - `atomic {S}`
- Retry: block until alternate path possible
  - `atomic {... retry;...}`
- Orelse: compose alternate atomic blocks
  - `atomic {S1} orelse{S2} ... orelse{Sn}`
- Tryatomic: atomic with escape hatch
  - `tryatomic {S} catch(TxnFailed e) {...}
- When: conditionally atomic region
  - `when (condition) {S}`

Builds on prior research in other systems.
Transactional Java Platform → Java Platform

- Transactional Java Platform • Standard Java Platform+ STM API

```java
atomic {
    S;
}

while(true) {
    TxnHandle th = txnStart();
    try {
        S'
        break;
    } finally {
        if(!txnCommit(th))
            continue;
    }
}
```

- Transactional control flow modeled via:
  - txnStart and txnCommit
  - Loop (Repeat txn if commit fails)
  - Exception handling
    - Commit on normal / exceptional exit
    - STM conflicts trigger exceptions
    - Repeat if commit fails
Transactional Java Platform → Java Platform

- Transactional Java platform
- Standard Java platform + STM API

```java
atomic {
    tmp = a.x;
    tmp = foo(tmp);
    b.y = tmp;
}
```

- STM insert barriers on memory accesses
- Tracks accesses and detects conflicts
- Modifies functions calls

```java
while(true) {
    TxnHandle th = txnStart();
    try {
        tmp = stmRd(a.x);
        tmp = foo(#tx#)(a.x);
        stmWr(b.y, tmp);
        break;
    } finally {
        if(!txnCommit(th))
            continue;
    }
}
```
VM Implementation

- Handling Function Calls
  - On Demand Cloning
  - Duplicate Virtual Table Entries

- Transaction Record
  - Per Object or Word
  - Pointer Sized

- Common Runtime Infrastructure
  - McRT Runtime
Leveraging the JIT – STM JIT

- Identifies transactional regions in Java+STM code
- Differentiates top-level and nested transactions
- Inserts read/write barriers in transactional code
- Maps STM API to first class opcodes in JIT IR
- Performs Optimizations
  - Standard Optimizations: CSE
  - STM Specific: RW Barrier Elimination, Partial Inlining, Runtime Filtering

Intel’s Java STM extends Java technology & utilizes VM/JIT for optimization
Agenda

➤ Introduction to Concurrency Transactional Memory
➤ Software Transactional Memory
➤ Hardware Transactional Memory
  • HTM and Software Usage
  • Java Technology-based Application Characteristics
  • Software Lock Elision
  • Speculative Execution
➤ Results
➤ Summary & Call to Action
Hardware Transactional Memory

- Hardware views individual threads as executing reads/writes to shared memory
- In a transaction we want the series of reads/writes to occur at a single instant in time
- Idea: Utilize Caches/Coherence in Modern Microprocessors
  - Coherence Lookup detects conflicts between transactions
  - Store the Undo logs in the HW Buffers for “rollback”
  - Several Implementations
- If Hardware is built
  - Can existing software use it?
  - Can new software use it?
Existing Software Usage

- The Industry has invested in the Existing Programming Model for Java technology
- Changing Programming Model is hard
- Enabling new hardware is challenging
- Improving Performance of Legacy (Java) Software on Hardware Transactional Memory is important
Java Application Characteristics - I

```java
//addElement inserts an item into the next available entry in an array
public void addElement(int value) {
    int indexRelativeToCache = m_firstFree - m_buildCacheStartIndex;
    if (indexRelativeToCache >= 0 && indexRelativeToCache < m_blocksize) {
        m_buildCache[indexRelativeToCache] = value;
        ++m_firstFree;
    } else {
        int index = m_firstFree >>> m_SHIFT;
        int offset = m_firstFree & m_MASK;
        if (index >= m_map.length) {
            int newsize = index + m_numblocks;
            int[][] newMap = new int[newsize][];
            System.arraycopy(m_map, 0, newMap, 0, m_map.length);
            m_map = newMap;
        }
        int[] block = m_map[index];
        if (null == block)
            block = m_map[index] = new int[m_blocksize];
        block[offset] = value;
        m_buildCache = block;
        m_buildCacheStartIndex = m_firstFree - offset;
        ++m_firstFree;
    }
}
```

Hot Paths have lot fewer dynamic instructions
Java Application Characteristics - II

```java
public synchronized int getItemId() {
    int temp = itemId;
    return temp;
}
```

T1:
- o.getItemId();
- Acquire Object o lock
- Read itemId
- Release Object o lock
- return

T2:
- o.getItemId();
- Wait for Object o lock
  :  
- Acquire Object o lock
- Read itemId
- Release Object o lock

Java technology-based Server Workload

Java platform locks optimized for
- No-contention case
- But not optimized for the case of `lock contention with no real data conflicts`

Loss of Concurrency due to Contended Locks
Software Usage of HTM

A Java based system that utilizes HW Transactional Memory (“isolation” & “atomicity”) with no changes to Software

- Software Lock Elision – Improve Concurrency by speculatively executing locking code and roll back on actual data contention
- Speculative Execution – Improve Single thread performance by speculatively executing hot paths and roll back on assumption violation

**SE Example**

```
T1: begin_trans
    o.getItemId();
end_trans
```

**SLE Example**

```
T1: begin_trans
    Read itemId
end_trans
    return

T2: begin_trans
    Read itemId
end_trans
    return
```
Agenda

➤ Introduction to Concurrency Transactional Memory
➤ Software Transactional Memory
➤ Hardware Transactional Memory
➤ Results
  • Software Transactional Memory
  • Software usage of Hardware Transactional Memory
➤ Summary & Call to Action
STM Results

- Implemented a Java based STM System with best effort optimization (see *PPoPP 2008 paper for more details*)
- Experience:
  - Smaller workloads easier to transactionalize, Larger workloads harder due to I/O, Native Code.
  - Weak Atomicity STMs have subtle semantic challenges.
- Performance:
  - Single Thread overhead about 2x from Software STM over lock based
  - Good scalability under lower contention and for micro benchmarks (HashMap)
Software Usage of HTM Results

- Demonstrated transparent (to programmer) usage of HTM with actual Java technology-based prototypes (see ISCA 2007 paper for more details)
- Both Software Lock Elision and Speculative Execution prototypes run realistic workloads using future proposed instructions.
- Speculative Execution demonstrates about 10% performance improvement for DaCapo workload suite
Summary

- TM is a tool for Concurrency Control in Multi-Core Systems
- STM is a software technology for developing *new* parallel software
- HTM is a hardware technology that both *existing* and *new* software can utilize to improve *performance* as well as *concurrency*
- Intel has *developed prototypes* for both Java based STM’s and transparent acceleration of Java technology-based software for HTMs
For More Information

➢ Software
  • Experimental Intel STM & Multicore Software http://whatif.intel.com
    - C/C++ compiler
    - STM run time library
  • RSTM: http://www.cs.rochester.edu/research/synchronization/rstm
  • Stanford STAMP: http://stamp.stanford.edu

➢ Tutorials/Conferences
  • PACT Tutorial http://csl.stanford.edu/~christos/
  • Micro 2007 Tutorial – Dan Grossman, Suresh Srinivas, Craig Zilles, Vijay Menon
  • TRANSACT Conferences

➢ Sites:
  • http://en.wikipedia.org/wiki/Software_transactional_memory
  • http://research.sun.com/scalable/
For More Information

Papers:

- Brevnov, Dolgov, Kuznetsov, Yershov, Shakin, Chen, Menon, Srinivas, Practical Experiences with Java* Software Transactional Memory, PPoPP’08
- Neelakantam, Rajwar, Srinivas, Srinivasan, Zilles, Hardware Atomicity for Reliable Software Speculation, ISCA’07
- Shpeisman, Menon, Adl-Tabatabai, Balensiefer, Grossman, Hudson, Moore, Saha, Enforcing Isolation and Ordering in STM, PLDI’07
- Saha, Adl-Tabatabai, Hudson, Minh, Hertzberg, McRT-STM: A High Performance Software Transactional Memory System for a Multi-...
THANK YOU

Suresh Srinivas, Principal Engineer,
Vyacheslav Shakin, SW Engineering Manager,
Software & Solutions Group,
Intel

TS 6316