Hardware Transactional Memory with Operating System support

Saša Tomić   Adrián Cristal   Osman Unsal   Mateo Valero

Barcelona Supercomputing Center,
Universitat Politecnica de Catalunya,
Spain

Workshop on Highly Parallel Processing on a Chip (HPPC),
August 28, 2007, IRISA, Rennes, France
# Introduction

- Transactional Memory Introduction
- Previous Work

# Our Contribution

- Goal
- Basic Ideas for the Implementation
- Software modifications
- Hardware modifications
- Non-conflicting Example
- Conflicting Example
- Commit and Abort
But, locks are working OK! Or are they?

- We are moving from multicore (2, 4, 16 cores) to manycore (128, 1024 - exists: SGI® Altix® 4700, 4096 cores, ... )!
- It is very difficult to use locks for synchronization, for large programs
  - deadlock, race condition, lock contention, composability, priority inversion, convoying, debugging
But, locks are working OK! Or are they?

- We are moving from multicore (2, 4, 16 cores) to manycore (128, 1024 - exists: SGI® Altix® 4700, 4096 cores, ... )!
- It is very difficult to use locks for synchronization, for large programs
  - deadlock, race condition, lock contention, composability, priority inversion, convoying, debugging
Are there any alternatives to the locks?

Lock-free programming:

- Just put *atomic* around the block of a code we want to protect and the framework will (should) deal with the concurrency
  - Software TM (STM): library/compiler tracks every read/write inside atomic segment and ensures the correctness
  - Hardware TM (HTM): processors/specialized hardware ensures correctness
  - Hybrid TM (HyTM): the mix of STM and HTM
Are there any alternatives to the locks?

Lock-free programming:

- Just put *atomic* around the block of a code we want to protect and the framework will (should) deal with the concurrency
  - Software TM (STM): library/compiler tracks every read/write inside atomic segment and ensures the correctness
  - Hardware TM (HTM): processors/specialized hardware ensures correctness
  - Hybrid TM (HyTM): the mix of STM and HTM
Introduction

1. Transactional Memory Introduction
2. Previous Work

Our Contribution

2. Goal
3. Basic Ideas for the Implementation
4. Software modifications
5. Hardware modifications
6. Non-conflicting Example
7. Conflicting Example
8. Commit and Abort
TCC [ISCA’04], LogTM [HPCA’06]
- TCC: buffer all writes in L1 until commit
- LogTM: write in-place and prevent other threads from reading/writing speculative data
- TCC: faster [ISCA’07], limited transaction size

UTM [HPCA’05], VTM [ISCA’05]
- Unbounded, good performance, but expensive, extra hardware mostly idle

PTM [ASPLOS’06], XTM [ASPLOS’06]
- Start the execution in the HTM-enabled L1 cache
- restart and re-execute the transaction with the other mechanism on an interrupt, context switch, overflow, etc.
Introduction

Transactional Memory Introduction
Previous Work

Our Contribution

Goal
Basic Ideas for the Implementation
Software modifications
Hardware modifications
Non-conflicting Example
Conflicting Example
Commit and Abort
Don’t go that low

- Create the HTM that **does not overflow**
- Minimize the amount of useless work - always work the same way
- Instead of modifying the L1 cache, **modify the TLB**
- Pay the price **only** on the **first** transactional access to the page
- Do it simply and get high execution speed
Basic Ideas for the Implementation

Outline

1. Introduction
   - Transactional Memory Introduction
   - Previous Work

2. Our Contribution
   - Goal
   - Basic Ideas for the Implementation
     - Software modifications
     - Hardware modifications
     - Non-conflicting Example
     - Conflicting Example
     - Commit and Abort
Basic Ideas for the Implementation

The Idea

- One extra copy of the data, per transaction, is sufficient
- Create a copy of the cacheline on the first transactional write to it
- Conflict detection - per cacheline; bookkeeping - per page
- Use the existing Virtual Memory mechanisms for accessing both copies of the cacheline
- The processor knows if it wants a primary or secondary copy of the data
1. **Introduction**
   - Transactional Memory Introduction
   - Previous Work

2. **Our Contribution**
   - Goal
   - Basic Ideas for the Implementation
   - **Software modifications**
   - Hardware modifications
   - Non-conflicting Example
   - Conflicting Example
   - Commit and Abort
Operating System modifications

- Manage TState table - transactional state of every processor
- Manage an additional VPT for secondary pages, and give a pointer to the processor (on request)
- Iterate the secondary VPT on transaction abort or commit
- Manage the space for conflict detection information
- All data is stored in cacheable, shared, physical memory, accessible from all processors
- Manage the context switch, process migration etc. (for details see the paper)
Outline

1. Introduction
   - Transactional Memory Introduction
   - Previous Work

2. Our Contribution
   - Goal
   - Basic Ideas for the Implementation
   - Software modifications
   - Hardware modifications
   - Non-conflicting Example
   - Conflicting Example
   - Commit and Abort
TLB entry modification - Overview

- T bit differentiates between the secondary (backup) and primary address of the page
- TBp is a pointer to the address of the transactional status bitmap (TB)
- TBp can be zero, then the page is not transactional
- TB holds the transactional information for the page (used for conflict detection)
Hardware modifications

**TLB entry modification - schematic**

- \( bl \) - cacheline block number inside page (lower 6..11 bits of the address)
- The additional hardware is simple
- This is the implementation *per TLB entry*
Outline

1. Introduction
   - Transactional Memory Introduction
   - Previous Work

2. Our Contribution
   - Goal
   - Basic Ideas for the Implementation
   - Software modifications
   - Hardware modifications
   - Non-conflicting Example
   - Conflicting Example
   - Commit and Abort
Transaction READ is attempted
TB update - illustration

- OS allocates the TB and gives the processor a pointer to it
- Processor updates the TB for the page
Non-conflicting Example

TB update - illustration

Transactional WRITE is attempted
- OS allocates a secondary page (private for transaction)
- Processor creates a copy of the block, updates the TB
Normal reads and writes will consult the TB, if it exist

Strong atomicity supported
Outline

1. Introduction
   - Transactional Memory Introduction
   - Previous Work

2. Our Contribution
   - Goal
   - Basic Ideas for the Implementation
   - Software modifications
   - Hardware modifications
   - Non-conflicting Example
   - Conflicting Example
   - Commit and Abort
Every processor does conflict detection (for scalability)
E.g. transactional READ by processor 4
Conflicting Example

Conflict detection - illustration

- There is no conflict on multiple readers
- Processor updates the TB for the page
**Transitional WRITE by the processor 1**

**Conflict with processor 4**
Conflict detection - illustration

- Processor 1 sends a kill request to processor 4
- Processor 4 restores cachelines and clears TXID
- OS allocates a secondary page (private for transaction)
- Copy the block to the secondary page
Conflicting Example

Conflict detection - illustration

- Processor updates the TB for the page
- Transaction proceeds
Outline

1. Introduction
   - Transactional Memory Introduction
   - Previous Work

2. Our Contribution
   - Goal
   - Basic Ideas for the Implementation
   - Software modifications
   - Hardware modifications
   - Non-conflicting Example
   - Conflicting Example
   - Commit and Abort
Commit, Abort

- On **commit**, the TB bits are reset for this TXID, for every page touched by the transaction.
- On **abort**, the TB bits are reset for this TXID, for every page touched by the transaction, and cachelines are copied from the secondary to the primary page.
**Summary**

- Memory is cheap, do not try to save it, as this would probably increase the transaction execution time *and* the hardware complexity.

- Hardware-only, as well as Software-only solutions are not the answer. The right place is *probably* somewhere in between.

**Outlook**

- We have implemented a proof-of-concept version in the PTLsim, x86_64 full-system simulator, from the University of New York at Binghamton.
- Experimenting with various configurations and applications is on top of the TODO list.
Questions?

Saša Tomić
sasa.tomic@bsc.es