Design and Use of Transactional Memory in MPSoCs

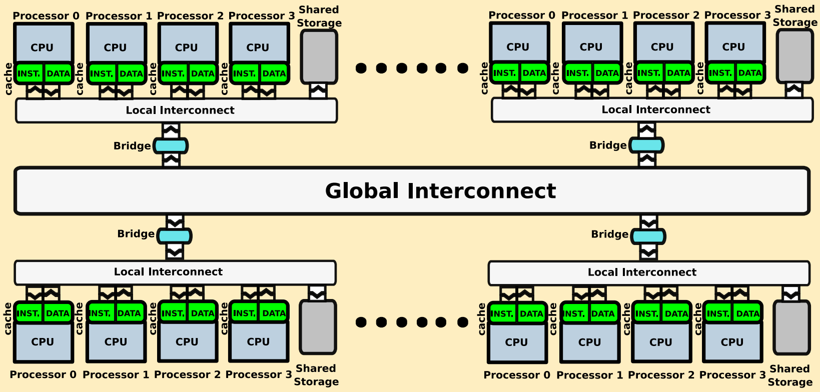
Frédéric Pétrot
Quentin Meunier

System-Level Synthesis Group
TIMA Laboratory
46, Av Félix Viallet, 38031 Grenoble, France

MPSoC'09

Context: Foreseeable architectural template

Logicically shared, physically distributed memory architecture



- Non-uniform memory access times
- Caches for programming simplicity
- Coherent memory

Context: Efficient exploitation of the available parallelism

Few programs written to exploit parallelism effectively

- Often limited to large parallel workloads
- But may change with the generalization of multi-core PCs

Popular programming model: Threads

- Coordination of execution:
  - Spin Locks
  - Mutexes, Semaphores, Read/Write Locks, Barriers, ...
  - Condition

Limits

- Experience shows that these programs are difficult to:
  - Design, Implement, Debug, Maintain
- ...and often do not perform or scale well

$\longrightarrow$ Need for other programming constructs

Outline

Outline

Transactional Memory (TM)

Transaction API

- Several queries must *appear* as to execute atomically

```
begin_transaction();
 /* All actions taking place here occur in Atomicity
  * and in Isolation */
end_transaction();
```

TM Programming Model ensures

- Atomicity:
  Intermediate state of the transaction hidden from the perspective of other processors
- Isolation:
  Concurrent executing threads cannot interfere with the executing transaction

Example

Thread t1 :

Example Description with Locks

- Several shared structures, one lock per structure
- Modify a structure s1:

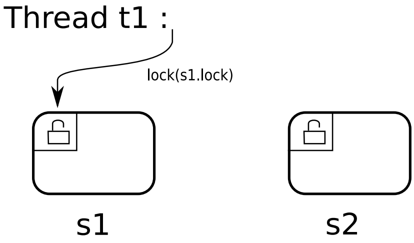
```
lock(s1.lock);
    // modify s1 fields
unlock(s1.lock);
```

Example

Example Description with Locks

- Several shared structures, one lock per structure
- Modify a structure s1:

```
lock(s1.lock);
    // modify s1 fields
unlock(s1.lock);
```

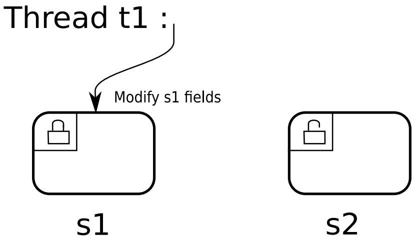


Example

Example Description with Locks

- Several shared structures, one lock per structure
- Modify a structure s1:

```
lock(s1.lock);
   // modify s1 fields
unlock(s1.lock);
```

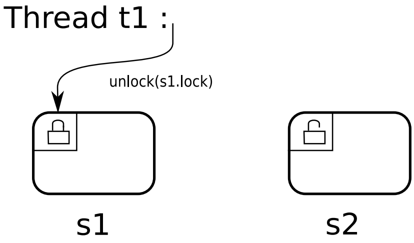


Example

Example Description with Locks

- Several shared structures, one lock per structure
- Modify a structure s1:

```
lock(s1.lock);
    // modify s1 fields
unlock(s1.lock);
```



Example

Example Description with Locks

- Several shared structures, one lock per structure
- Modify a structure s1:

```
lock(s1.lock);
    // modify s1 fields
unlock(s1.lock);
```

- Now suppose we want to do atomic operations between two objects
    - Risk deadlock (or impose a total order on structures)
    - Or requires additional locks on tuples of objects
      ⇒ New interface

Thread t1 :

Example

Example Description with Locks

- Several shared structures, one lock per structure
- Modify a structure s1:

```
lock(s1.lock);
    // modify s1 fields
unlock(s1.lock);
```

- Now suppose we want to do atomic operations between two objects
  - Risk deadlock (or impose a total order on structures)
  - Or requires additional locks on tuples of objects
    $\Rightarrow$ New interface

Thread t1 :

With Transactions

Example

Example Description with Locks

- Several shared structures, one lock per structure
- Modify a structure s1:

```
lock(s1.lock);
    // modify s1 fields
unlock(s1.lock);
```

- Now suppose we want to do atomic operations between two objects
  - Risk deadlock (or impose a total order on structures)
  - Or requires additional locks on tuples of objects
    $\Rightarrow$ New interface

Thread t1 :

With Transactions

- Modify a structure:

```
begin_transaction();
    // modify s1 fields
end_transaction();
```

Example

Thread t1 :

Example Description with Locks

- Several shared structures, one lock per structure
- Modify a structure s1:

```
lock(s1.lock);
   // modify s1 fields
unlock(s1.lock);
```

- Now suppose we want to do atomic operations between two objects
  - Risk deadlock (or impose a total order on structures)
  - Or requires additional locks on tuples of objects
    $\Rightarrow$ New interface

With Transactions

- Modify a structure:

```
begin_transaction();
   // modify s1 fields
end_transaction();
```

- Modify two structures atomically:

```
begin_transaction();
   // modify s1 fields
   // modify s2 fields
end_transaction();
```

Types of Transactional Memories

Software Transactional Memory (STM)

- Limited hardware support required: only atomic operations
- Many do not believe in STM, controversial subject:
  Software transactional memory: why is it only a research toy? [CBM+08]

Hardware Transactional Memory (HTM)

- Specific support to transactions in hardware
  requires modifications of the whole memory hierarchy [HM93]
- No existing machine currently provides such a support
  Sun Microsystems *Rock* multicore was said to be canceled June 15th, 2009[a]

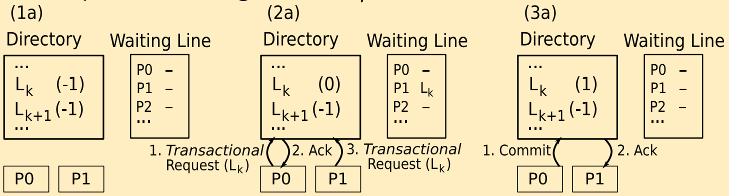
[a]Sun did not confirm or infirm officially

HTM Systems

General Characteristics & Problems relative to HTM Systems

- Granularity of accesses: cache line
- Need to detected conflicting accesses to a variable:

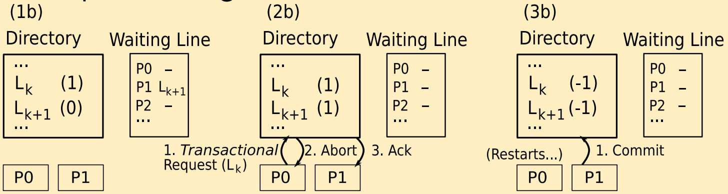
  ⇒ Requires tracking the read/write accesses to a line



HTM Systems

General Characteristics & Problems relative to HTM Systems

- Transaction can abort if there is a conflict:
  i.e. 2 transactions on same line including a write
  ⇒ Requires storing both **old** and **new** values



  - Speculated data (Data that is *computed* but not yet *committed* to memory) have to be stored somewhere
    ⇒ HTM sets can overflow (finite capacity)
- Not so simple architectural support within memory and caches
- Cache-coherence protocol dependent

Very challenging to define and build a working system

Classification of HTM Systems

Main Criteria

- Conflict Detection: when to detect conflicts?
  - Eager: as soon as two concurrent transactions attempt to access the same line
  - Lazy: at commit time
- Version Management: where to store old and new values?
  - Eager: Store the new values in place and the old ones in a log
    Fast commit
  - Lazy: Leaves old values in memory and log the new ones
    Fast abort
- Conflict Resolution: what to do when a conflict is detected?
  - Eager: Stall/Abort the requester(s)
    $\Rightarrow$ Stalling the requester also requires to be able to break potential deadlock cycles by making some processors abort
  - Lazy: Abort the committer

Main Existing HTM Systems Implementations

Main Existing HTM Systems

Short Name	Full Name	Reference
LogTM	Log Based Transactional Memory[a]	[MBM$^+$06a]
TCC	Transactional Coherence and Consistency	[HCW$^+$04]
VTM	Virtualizing Transactional Memory	[RHL05]
UTM	Unbounded Transactional Memory	[AAK$^+$05]
LTM	Large Transactional Memory	[AAK$^+$05]
Bulk	-	[CTTC06]

[a] and its variants: LogTM-SE [YBM$^+$07], TokenTM [BGH$^+$08] and LogTM-VSE [SVG$^+$08]

Standard Design Space Choices and Positioning

LL: Lazy Conflict Detection, Lazy Version Management, committer wins
EL: Eager Conflict Detection, Lazy Version Management, requester wins
EE: Eager Conflict Detection, Eager Version Management, requester stalls

LogTM	→	EE	TCC	→	LL	VTM	→	EL
UTM	→	EE	Bulk	→	LL	LTM	→	EL

Outline

Design Choices & Restrictions for MPSoC

Design Choices: Simplicity

- Use of simple RISC processors, e.g. Sparc V8, Mips 4K
- Write-through, Direct-mapped caches
- Physical address space (no MMU)

Other Design Choices: Still simplicity

- Eager Conflict Detection, Eager Version Management, Resolution scheme based on stalling the requester
- Write-Through Invalidate cache coherence protocol
- Flat transaction nesting semantic

Restrictions: Always simplicity

- One thread per processor, each thread being pinned on a processor
- OS calls and I/O accesses forbidden inside transactions

Architecture and OS Modifications

Architecture Modifications

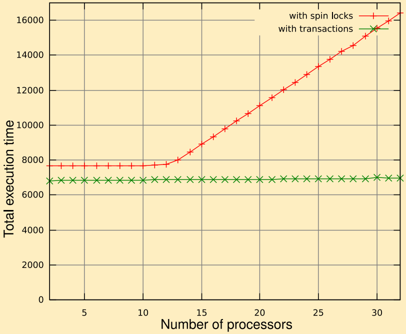
- API
  - `begin_transaction()` and `end_transaction()`: Processor configuration to perform transactional accesses
  - `store_log_address(address)`: Configuration of a specific register in cache
- On processors
  - Addition of a shadow register file in processor, for aborts
  - New instructions or new semantics on existing ones: e.g. `rdasr` on Sparc or `mtc0` on Mips32
- On caches and memories
  - Read and Write sets tracked with R/W bits associated to each cache and memory line
  - Value log: Data Cache
  - Additions to the cache coherency protocol and the interconnect protocol

Status

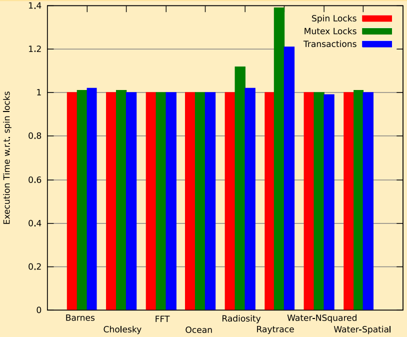
LightTM: HTM System under design

- Cycle Accurate Bit Accurate modeling of the whole system
- 2-32 Sparc V8, Abstract NoC Interconnect, modified SoCLib models
- Kernels and Splash 2 (parallel workloads) benchmarks

Preliminary results



Run times vs Number of processors for Kernel



Run times for Splash-2 benchmarks, 16 procs

Wrap-up

Transactional Memory

- Concept introduced more than 15 years ago[HM93]
- Currently generating a lot of research
  In Computer Architecture and also Parallel Programming conferences
- Proved to be pretty efficient
- Also proved to be quite complex and costly!

Is it useful for MPSoC?

- More and more programmable IPs in High End Consumer MPSoC
- Need of a simple shared memory parallel programming paradigm
- But quite complex to implement and not RT friendly

May be (part of) a solution

DATE 10
Design, Automation & Test in Europe
8-12 March, 2010 - Dresden, Germany
The European Event for Electronic
System Design & Test

A8: Multi-core Platforms

Design and Use of Transactional Memory in MPSoCs

Frédéric Pétrot
Quentin Meunier

System-Level Synthesis Group
TIMA Laboratory
46, Av Félix Viallet, 38031 Grenoble, France

MPSoC'09

References I

C. Scott Ananian, Krste Asanovic, Bradley C. Kuszmaul, Charles E. Leiserson, and Sean Lie, *Unbounded transactional memory*, Proc. 11th International Conference on High-Performance Computer Architecture (11th HPCA'05) (San Francisco, CA, USA), IEEE Computer Society, February 2005, pp. 316–327.

Jayaram Bobba, Neelam Goyal, Mark D. Hill, Michael M. Swift, and David A. Wood, *TokenTM: Efficient execution of large transactions with hardware transactional memory*, Proc. 35th International Symposium on Computer Architecture (35th ISCA'08) (Beijing), ACM SIGARCH, June 2008.

References II

Jayaram Bobba, Kevin E. Moore, Haris Volos, Luke Yen, Mark D. Hill, Michael M. Swift, and David A. Wood, *Performance pathologies in hardware transactional memory*, Proc. 34th International Symposium on Computer Architecture (34th ISCA'07) (San Diego, California, USA), ACM SIGARCH, June 2007, pp. 81–91.

Calin Cascaval, Colin Blundell, Maged Michael, Harold W. Cain, Peng Wu, Stefanie Chiras, and Siddhartha Chatterjee, *Software transactional memory: why is it only a research toy?*, ACM Queue: Tomorrow's Computing Today **6** (2008), no. 5, 46–58.

Luis Ceze, James Tuck, Josep Torrellas, and Calin Cascaval, *Bulk disambiguation of speculative threads in multiprocessors*, ISCA, IEEE Computer Society, 2006, pp. 227–238.

References III

- Fraser and Harris, *Concurrent programming without locks*, ACMTCS: ACM Transactions on Computer Systems **25** (2007).
- Lance Hammond, Brian D. Carlstrom, Vicky Wong, Michael K. Chen, Christos Kozyrakis, and Kunle Olukotun, *Transactional coherence and consistency: Simplifying parallel hardware and software*, IEEE Micro **24** (2004), no. 6, 92–103.
- Maurice Herlihy and J. Eliot B. Moss, *Transactional memory: Architectural support for lock-free data structures*, ISCA, 1993, pp. 289–300.

References IV

Sudheendra Hangal, Durgam Vahia, Chaiyasit Manovit, Juin-Yeu Joseph Lu, and Sridhar Narayanan, *TSOtool: A program for verifying memory systems using the memory consistency model*, Proc. 31th Ann. Intl Symp. on Computer Architecture (31th ISCA'04), ACM Computer Architecture News (CAN) (Munich, Germany), ACM SIGARCH / IEEE Computer Society, June 2004, Published as Proc. 31th Ann. Intl Symp. on Computer Architecture (31th ISCA'04), ACM Computer Architecture News (CAN), volume 32, number ?, pp. 114–123.

Kevin E. Moore, Jayaram Bobba, Michelle J. Moravan, Mark D. Hill, and David A. Wood, *LogTM: Log-based transactional memory*, Proceedings of the 12th International Symposium on High-Performance Computer Architecture, IEEE Computer Society, February 2006, pp. 254–265.

References V

Michelle J. Moravan, Jayaram Bobba, Kevin E. Moore, Luke Yen, Mark D. Hill, Ben Liblit, Michael M. Swift, and David A. Wood, *Supporting nested transactional memory in logTM*, Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2006, San Jose, CA, USA, October 21-25, 2006 (John Paul Shen and Margaret Martonosi, eds.), ACM, 2006, pp. 359–370.

Chaiyasit Manovit, Sudheendra Hangal, Hassan Chafi, Austen McDonald, Christos Kozyrakis, and Kunle Olukotun, *Testing implementations of transactional memory*, Proceedings of the 15th International Conference on Parallel Architecture and Compilation Techniques (15th PACT'06) (Seattle, Washington, USA) (Erik Altman, Kevin Skadron, and Benjamin G. Zorn, eds.), ACM, September 2006, pp. 134–143.

References VI

- Ravi Rajwar, Maurice Herlihy, and Konrad K. Lai, *Virtualizing transactional memory*, ISCA, IEEE Computer Society, 2005, pp. 494–505.
- Bratin Saha, Ali-Reza Adl-Tabatabai, and Quinn Jacobson, *Architectural support for software transactional memory*, MICRO, IEEE Computer Society, 2006, pp. 185–196.
- Daniel J. Sorin, Manoj Plakal, Mark D. Hill, Anne E. Condon, Milo M. Martin, and David A. Wood, *Specifying and verifying a broadcast and a multicast snooping cache coherence protocol*, Technical Report 1412, Univ. of Wisconsin Computer Sciences, Madison, WI, March 2000.
- M.M. Swift, H. Volos, N. Goyal, L. Yen, M.D. Hill, and D.A. Wood, *OS Support for Virtualizing Hardware Transactional Memory*, 2008.

References VII

Sasa Tomic, Adrián Cristal, Osman S. Unsal, and Mateo Valero, *Hardware transactional memory with operating system support, HTMOS*, Euro-Par Workshops (Luc Bougé, Martti Forsell, Jesper Larsson Träff, Achim Streit, Wolfgang Ziegler, Michael Alexander, and Stephen Childs, eds.), Lecture Notes in Computer Science, vol. 4854, Springer, 2007, pp. 8–17.

Luke Yen, Jayaram Bobba, Michael R. Marty, Kevin E. Moore, Haris Volos, Mark D. Hill, Michael M. Swift, and David A. Wood, *LogTM-SE: Decoupling hardware transactional memory from caches*, HPCA, IEEE Computer Society, 2007, pp. 261–272.