

# Hardware Transactional Memory with Operating System support

Saša Tomić   Adrián Cristal   Osman Unsal   Mateo Valero

Barcelona Supercomputing Center,  
Universitat Politècnica de Catalunya,  
Spain

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



















# 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

# 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

# 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

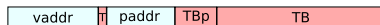
# Operating System modifications

- 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

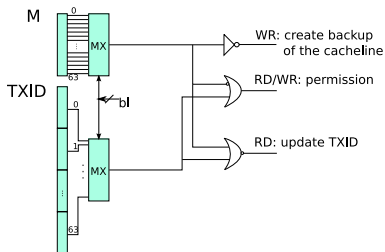
- 1 Introduction
  - Transactional Memory Introduction
  - Previous Work
- 2 Our Contribution
  - Goal
  - Basic Ideas for the Implementation
  - Software modifications
  - **Hardware modifications**
  - Non-conflicting Example
  - Conflicting Example
  - Commit and Abort

# 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)

# TLB entry modification - schematic

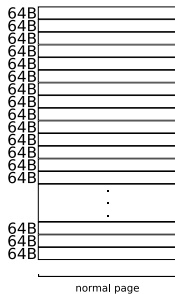


- **bl** - cacheline block number inside page (lower 6..11 bits of the address)
- The additional hardware is simple
- This is the implementation *per TLB entry*

# 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

# TB update - illustration

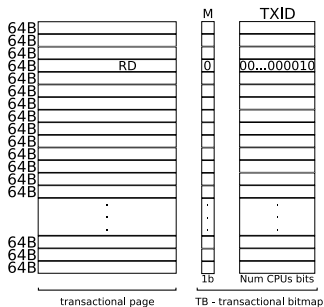


TX RD by CPU 1

- Transactional READ is attempted

## Non-conflicting Example

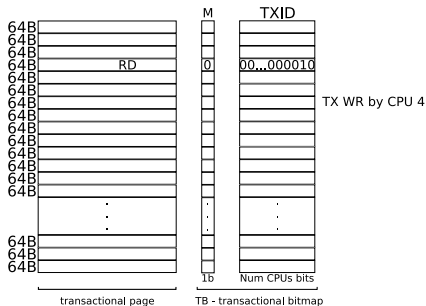
## TB update - illustration



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

Non-conflicting Example

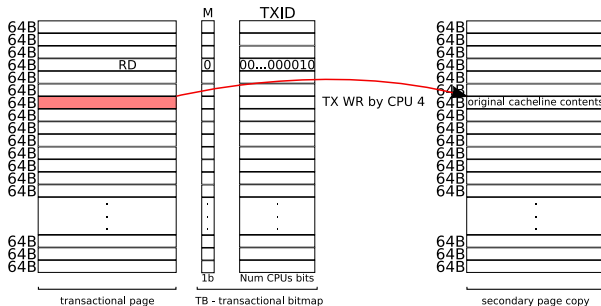
# TB update - illustration



- Transactional WRITE is attempted

## Non-conflicting Example

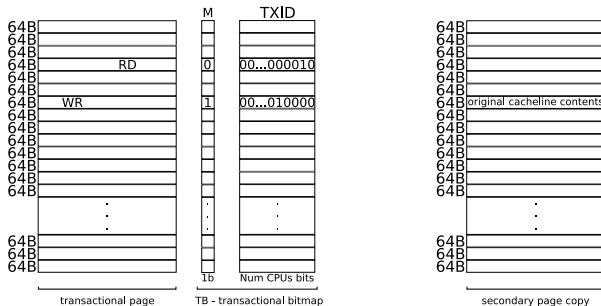
## TB update - illustration



- OS allocates a secondary page (private for transaction)
- Processor creates a copy of the block, updates the TB

## Non-conflicting Example

## TB update - illustration

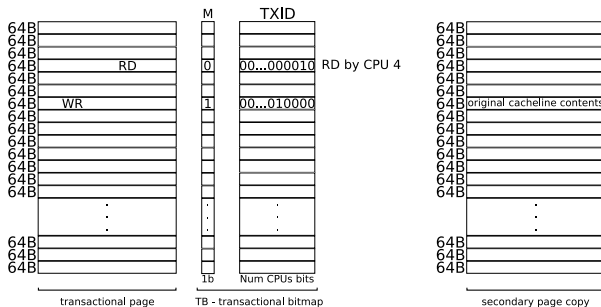


- Normal reads and writes will consult the TB, if it exist
- Strong atomicity supported

# 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

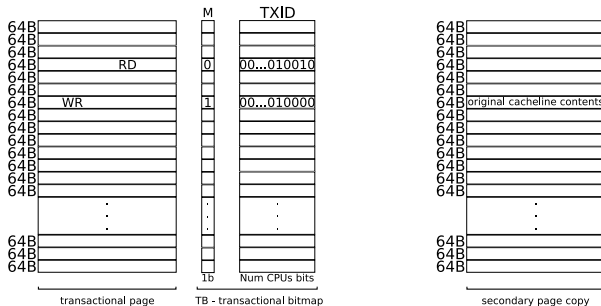
# Conflict detection - illustration



- Every processor does conflict detection (for scalability)
- E.g. transactional READ by processor 4

## Conflicting Example

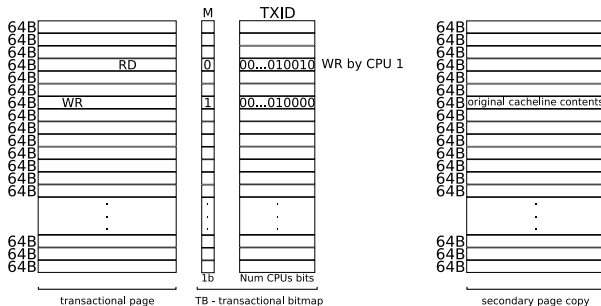
## Conflict detection - illustration



- There is no conflict on multiple readers
- Processor updates the TB for the page

## Conflicting Example

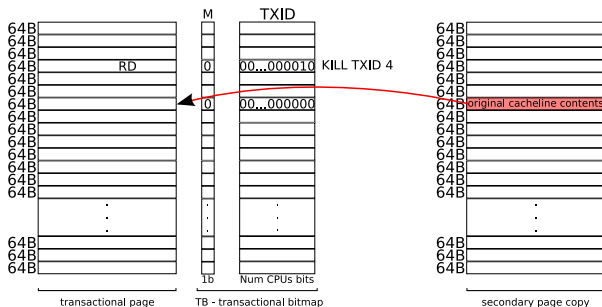
## Conflict detection - illustration



- Transactional WRITE by the processor 1
- Conflict with processor 4

## Conflicting Example

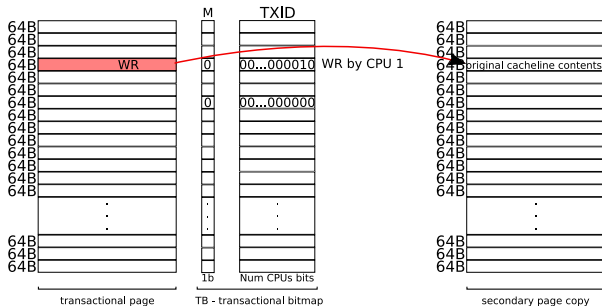
## Conflict detection - illustration



- Processor 1 sends a kill request to processor 4
- Processor 4 restores cachelines and clears TXID

## Conflicting Example

## Conflict detection - illustration



- OS allocates a secondary page (private for transaction)
- Copy the block to the secondary page





# 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

