Enhancing Efficiency of Hybrid Transactional Memory via Dynamic Data Partitioning Schemes

Pedro Raminhas, Shady Issa, Paolo Romano

pedro.raminhas@tecnico.ulisboa.pt

INESC-ID/Instituto Superior Técnico, University of Lisbon
Motivation

- Multicore processors are a mainstream technology
  - HPC
  - Laptops
  - Smartphones

- Hard to develop code that takes advantage of multicore processors
Motivation

- Multicore processors are a mainstream technology
  - HPC
  - Laptops
  - Smartphones

- Hard to develop code that takes advantage of multicore processors
Background of TM

- Transactional Memory [ISCA’ 93] promises:
  - Ease of use
  - Performance equal or better than fine-grained locking
- Extension of multi-processors’ cache coherence protocols
  - Speculative run => apply the concept of transactions from databases to parallel code
  - The underlying system guarantees isolation and atomicity

Balance[client]++;

...
Background of TM

- Transactional Memory [ISCA’ 93] promises:
  - Ease of use
  - Performance equal or better than fine-grained locking
- Extension of multi-processors’ cache coherence protocols
  - Speculative run => apply the concept of transactions from databases to parallel code
  - The underlying system guarantees isolation and atomicity

```plaintext
Threads

Tx_begin()

...

Balance[client]++;

...

Tx_end()
```
Software Transactional Memory (STM)

• Implements TM abstraction via a software library
  ✓ No restriction on transactions’ read and write-set sizes
  ✓ Good Scalability at high thread counts
  ✗ High instrumentation costs due to software-based tracking of reads/writes

```c
#include STM_LIB
Tx_begin();

STM_WRITE(Balance[client]++);

Tx_end();
```
Hardware Transactional Memory

• Best effort nature:
  - Based on extension of the cache coherence mechanisms
  - Recently introduced both in high-end (e.g. IBM Power8) and commodity (Intel Haswell) CPUs

✓ No software instrumentation over reads and writes
✓ Faster than STM for read and write-sets that fit in hardware

✗ No progress guarantees:
  - Transactions can abort even when running solo due to, e.g., exceeding cache capacities, context switches, timer interrupts...
Fallback mechanism

• After a few attempts using HTM, the transaction is executed using a software synchronization mechanism:

- Single global lock
- Progress guarantee
- No parallelism (abort concurrent HW transactions)
Hybrid Transactional Memory

- Fallback to STM => Hybrid Transactional Memory (Hybrid TM)

✅ preserve concurrency when activating the fallback path

❌ Coordinating HTM and STM is not trivial

Threads

Tx_begin()
.
.
.
Balance[client]++;
.
.
Tx_end()

#include STM_LIB
Problem

• State of the art Hybrid TM is not competitive with HTM and STM [PACT’14]

Exceed HTM capacity by requiring HTM to store the information of addresses accessed by STM [SPAA’11]

Spurious aborts of non-conflicting HTM transactions whenever an update STM transaction commits [ASPLOS’11]
DMP in a nutshell

- Dynamic Memory Partitioning (DMP) relies on memory protection OS mechanism to enforce correctness between HTM and STM

<table>
<thead>
<tr>
<th>Conventional HyTM design</th>
<th>DMP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>High</strong> overhead in absence of conflict:</td>
<td><strong>Assume low</strong> frequency of contention between HTM and STM</td>
</tr>
<tr>
<td>1. Expensive instrumentation of HTM path, or</td>
<td><strong>High efficiency in absence of contention between HTM &amp; STM:</strong></td>
</tr>
<tr>
<td>2. Reliance on non-scalable STM</td>
<td>1. STM agnostic: can use scalable OREC-based schemes</td>
</tr>
<tr>
<td><strong>Relatively low</strong> cost incurred upon contention</td>
<td>2. No instrumentation of HTM</td>
</tr>
<tr>
<td></td>
<td><strong>Relatively higher</strong> overhead in absence of conflict</td>
</tr>
</tbody>
</table>
Key-property

• Several reference benchmarks present *partitionability* in their memory accesses [SPAA’ 08], i.e. transactions that access one partition are unlikely to access other partitions.

Use multiple concurrency control mechanisms/STMs
Key-property

• DMP-TM is designed to take advantage of this property and minimize contention between HTM and STM
• Each partition fits the characteristics of HTM and STM, respectively
DMP in a nutshell

• Using Operating system’s memory protection mechanism

• Is possible to revoke the access to certain pages to either HTM or STM

• Depending on the access performed (read or write)

• Conflicts are detected at page level by catching a Segmentation Fault (SIGSEGV) signal

Access Violation SIGSEGV!

Page X

STM

HTM
DMP - Architecture
DMP - HTM

- No instrumentation (reduces probability of exceeding cache capacity)
DMP - STM

- **TinySTM [PPoPP ’08]**
- **Very robust across heterogeneous workloads**
- **Scalable Orec-based STM**
- **Added instrumentation to synchronise with HTM**
DMP - Memory Manager
DMP - Memory Manager

Responsible for mapping the heap twice
\texttt{mmap()} system call
DMP - Read operations

mprotect() system call to change protection of page
DMP - Write Operations

mprotect() system call to change protection of page
DMP - Unix Handler

Signal Handler

- HTM receives a SIGSEGV signal whenever it tries to access a page which will cause a conflict
- Handler is responsible for either wait until the completion of STM or change page protection
Problems - STM reads inconsistent values

1. Read variable on Page X

STM
Problems - STM reads inconsistent values

1. Read variable on Page X
2. Update variable on Page X
Problems - STM reads inconsistent values

1. Read variable on Page X
2. Update variable on Page X
3. SIGSEGV Signal
Problems - STM reads inconsistent values

1. Read variable on Page X

2. Update variable on Page X

4. Restore Page protection and Update variable On Page X

5. Commits!
Problems - STM reads inconsistent values

1. Read variable on Page X
2. Update variable on Page X
3. Restore Page protection and Update variable On Page X
4. Commits!
5. Commits!

1. Read variable on Page X
2. Update variable on Page X

Inconsistent value
Problems - STM reads inconsistent values

- **Solution:** Use per-page variable *transition count*:

1. Read variable on Page X
2. Update variable on Page X
4. Restore Page protection and Update variable On Page X
   - Transition count [Page X] ++
5. Commits!
6. Detects change of transition count upon each read => abort!
Problems - Unnecessary ping-pongs

1. Write variable on Page X
2. Update variable on Page X
3. SIGSEGV Signal

HTM → Page X → STM

Page X
Problems - Unnecessary ping-pongs

1. Write variable on Page X
2. Update variable on Page X
3. SIGSEGV Signal
4. Restore Page protection and Update variable On Page X
5. Aborts
Problems - Unnecessary ping-pongs

1. Write variable on Page X
2. Update variable on Page X
3. SIGSEGV Signal
4. Restore Page protection and Update variable On Page X
5. Revoke protection to STM and Write variable on Page X
Problems - Unnecessary ping-pongs

1. Write variable on Page X
2. Update variable on Page X
5. Revoke protection to STM and Write variable on Page X
6. Aborts

HTM → Page X → STM
Problems - Unnecessary ping-pongs

- **Solution:** Use per-page variable *writer count*: Handler only restores when no STM writers active on the page

```
1. Write variable on Page X
2. Update variable on Page X
5. Revoke protection to STM and Write variable on Page X
6. Aborts
```

HTM → Page X → STM
DMP - Scheduler

- HTM
- Auto-tuner
- STM
- Scheduler
- Signal handler
- Memory manager

Connections:
- HTM to Auto-tuner
- Auto-tuner to Memory manager
- Memory manager to Scheduler
- Scheduler to HTM
- HTM to Signal handler
- Signal handler to access violations
- Access violations to Scheduler
- Scheduler to acquire access rights
- Acquire access rights to Memory manager
DMP - Scheduler

Scheduler automatically classifies transactions:

• **HTM friendly** (< 5% of Capacity aborts)

• **HTM non-friendly** (Capacity aborts)
DMP - Auto-tuner

Automatically detect non-partitionable workloads:

- If we spend more than 20% of the time issuing sys calls:

- Fallback to one of the backends
Evaluation

• Microbenchmarks
  • **Best-case Scenario:** Completely disjoint set of pages, both incurring heterogenous workloads
  • **Worst-case Scenario:** Only one page

• **STAMP**
  • Genome
  • Intruder

• **TPC-C**
Evaluation

- Baselines
  - DMP
  - DMP w/Auto-Tune module
  - HyNOrec
  - HyTinySTM
  - HyTL2
  - HTM-SGL
  - NOrec
  - TinySTM

\{ STM, HyTM, STM, \}
Evaluation

- IBM Power8
- 10 cores
- 8 Threads per core
Evaluation - Disjoint Microbenchmark

![Graph showing speedup with respect to TinySTM for different schemes under 80 threads and varying percentage of long transactions.](image)

- **dmp**: 7x TinySTM
- **dmp-tune**: 20x TinySTM
- **htm-sgl**: 10x TinySTM
- **norec**: 1x TinySTM
- **hytinystm**: 10x TinySTM
- **hynorec**: 10x TinySTM
- **tinystm**: 1x TinySTM
- **hytl2**: 1x TinySTM

% Long Transactions (1, 2, 5, 10, 15, 50, 90, 100)
Evaluation - Non-disjoint Microbenchmark

Auto-Tune allows to avoid overheads in unfavourable workloads

0.06 x Throughput of TinySTM

Due to ping-pong thus fallback to TinySTM
Evaluation - STAMP

Evaluation of Genome Throughput

- dmp
- dmp-tune
- htm-sgl
- hynorec

Evaluation of Intruder Throughput

- norec
- hytinystm
- tinystm
- hytl2

Throughput (10^5 Tx/s) vs. Number of threads

1.5x performance improvement
Evaluation - TPC-C

High partitionability

Low partitionability

Throughput (10^5 Tx/s)

Number of threads

2.4 x STM

Uses auto-tuner

0.125x less than TinySTM
Summary

• DMP design optimized for workloads with infrequent conflicts between transactions executing in HW and SW

• DMP uses OS memory protection mechanism to devise partitions where back-ends can execute without interference.

• Up to ~20x speedups compared to state of the art HTM, STM and HyTM
Thank you

https://github.com/pedroraminhas/DMP-TM
http://www.gsd.inesc-id.pt/~praminhas/
pedro.raminhas@tecnico.ulisboa.pt