Improving Commit Scalability in Lazy Hardware Transactional Memory

Anurag Negi*, Rubén Titos-Gil^, Manuel E. Acacio^, Jose M. Garcia^, Per Stenström*

*Chalmers University of Technology, Sweden
^Universidad de Murcia, Spain

Fourth Swedish Workshop on Multicore Computing (MCC) at Linköping University, 2011
Where does HTM fit in the big picture?

- Economics: Need for constant innovation and higher performance
- Single thread performance saturation
- More cores to cooperatively solve problems
- Shared memory systems
- Troubles with lock-based programming – productivity, safety
- TM: freedom from locks

HTM: Improve common case performance
HTM: Economy and Performance

Performance

Economy

Productivity

FGLocks

HTM

STM

HTM Challenges
• Manage design complexity
  • Utilize existing mechanisms better
  • Minimize changes required
• Improve performance
  • Go lazy !!
  • Yet avoid bulk communication !!!
Managing design complexity by utilizing existing mechanisms better.

Use coherence protocol to detect conflicts early and track these at cache line granularity.

Managing design complexity by minimizing changes.

No ad-hoc communication hardware for TM and piggy-back TM information on coherence messages.
Improving performance

Improving performance by going lazy

Optimistically *run past conflicts*
*Minimize abort overhead*
*Utilize MLP* better

Improving performance by avoiding bulk communication

Lightweight commits using *point-to-point messaging only between affected cores*
Scalability of lazy commits

Naïve: One at a time … the entire address space is **one giant bank**

Better: **Split** address space into **banks** … lock all required banks prior to committing updates … ensure progress guarantees

**Ideal:** Ensure **conflicting transactions re-execute** and prevent re-executions/new transactions **from reading locations not yet updated**
Prior Work

- EAZY-HTM[Micro2009]
  - Detect early – Resolve late
  - Ad-hoc communication channel for TM
  - Relies on directory communication for correctness

The correctness concern

Prevent other cores from accessing lines that are part of a committing transaction’s write-set but haven’t yet been made globally visible
The correctness concern in more detail

\[ \text{L1@Core1: } \{X_{\text{old}}, Y_{\text{old}}\} \]

\[ \text{TCommit@Core2: } \{X_{\text{new}}, Y_{\text{new}}\} \]

\[ \text{INV}(X) \]

\[ \text{L1@Core1: } \{Y_{\text{old}}\} \]

\[ \text{Core1:TRead}(X) \quad X_{\text{new}} \]

\[ \text{Core1:TRead}(Y) \quad Y_{\text{old}} \]

\[ \text{TCommit@Core1: } \{P, Q\} \]

\[ \text{INV}(Y) \]

\[ \text{L1@Core1: } \{\} \]

Core 1 commits an inconsistent computation

Atomicity requires Core 1 to either see \((X_{\text{old}}, Y_{\text{old}})\) or \((X_{\text{new}}, Y_{\text{new}})\) but not \((X_{\text{new}}, Y_{\text{old}})\)

The EAZY-HTM Approach

*Every first TRead or TWrite to a cache line communicates with the directory*

*Ensures correctness but causes severe performance degradation*
Reason for performance degradation

Most cache lines accessed in a typical transaction are not contended

Excessive communication with the directory causes congestion

The π-TM Approach

*Speed up the common case*

*Do extra work only for contended lines*
The π-TM Approach

Goals
- Speed up the common case
- Do extra work only for contended lines

Design changes
- Add π-bit to track contended lines
- Pessimitically Invalidate such lines on commit or abort

Other aspects
- No ad-hoc communication channel for TM
- TM info is piggy-backed on coherence messages
Incorporating adaptability

Why?
For short transactions with high contention, early conflict detection can increase transactional execution time

Lazy Detection and Resolution
Commit scalability problems but works well when application scalability is the dominant limiting factor (high contention)

We employ a global commit token (GCT) scheme in such scenarios
Each thread decides locally whether to use \( \pi \)-mode or GCT-mode
Both \( \pi \)-mode or GCT-mode transactions can coexist safely
Most applications run in \( \pi \)-mode
Estimating impact

Baseline
Faithfully implement Eazy-HTM information flow
However, we use the NoC for communication (no ad-hoc communication)
Coherence requests carry TM info as well

$\pi$-TM is implemented on top of this baseline
Adaptability mechanisms are enabled

Other configurations evaluated
EE: LogTM, an eager conflict resolution design
LL-GCT: Global commit token (transactions commit on at a time)
LL-STCC: A detailed scalable TCC implementation
Performance

Baseline

Effect of adaptability

Improved commit bandwidth

Best overall performance

4bars (L2R):
π-TM
EE(LogTM)
LL-GCT
STCC

16 threads on 16 cores, SIMICS+GEMS, STAMP applications
Conclusion

π-TM achieves the following:

A fully decentralized scalable commit protocol
Only conflicting threads/transactions get affected
Low design cost
Performs the best among evaluated design points