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

Outline

The importance of HTM

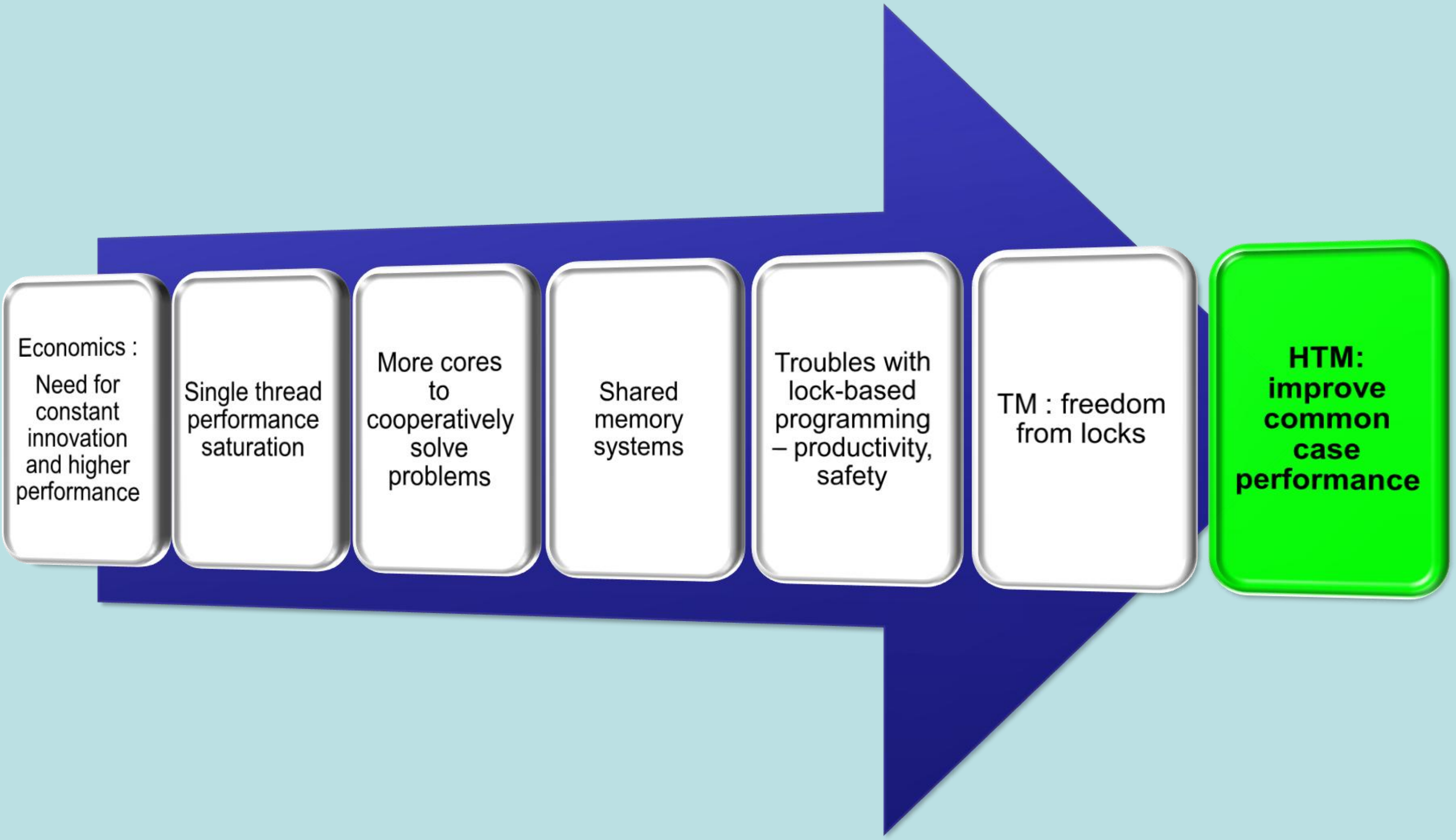
The key challenges

An approach to finding solutions

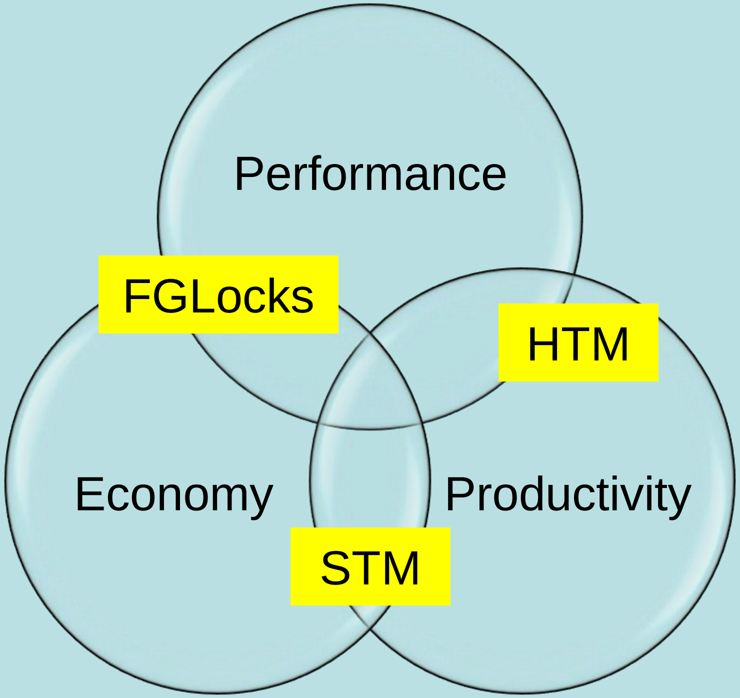
Prior work and associated inefficiencies

The π-TM approach

Where does HTM fit in the big picture?



HTM: Economy and Performance



HTM Challenges

- Manage design complexity
  - Utilize existing mechanisms better
  - Minimize changes required
- Improve performance
  - Go lazy !!
  - Yet avoid bulk communication !!!

Managing complexity

Managing design complexity by utilize 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 communcation hardware for TM

and

Piggy-back TM information ***on coherence messages***

Improving performance

Improving performance by going lazy

Optimisitically ***run past conflicts***

Minimize abort overhead

Utilize MLP better

Improving performance by avoiding bulk commuication

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

L1@Core1: {X_{old}, Y_{old}}

TCommit@Core2: {X_{new}, Y_{new}}

INV(X)

L1@Core1: {Y_{old}}

DELAY

Core1:TRead(X) ← X_{new}

Core1:TRead(Y) ← Y_{old}

TCommit@Core1: {P, Q}

INV(Y)

L1@Core1: {}

Core 1 commits an inconsistent computation

Atomicity requires Core1 to **either see (X_{old},Y_{old}) or (X_{new},Y_{new})** but **not (X_{new},Y_{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 **π-mode or GCT-mode**

Both π-mode or GCT-mode transactions can coexist safely

Most applications run in π-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

π-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

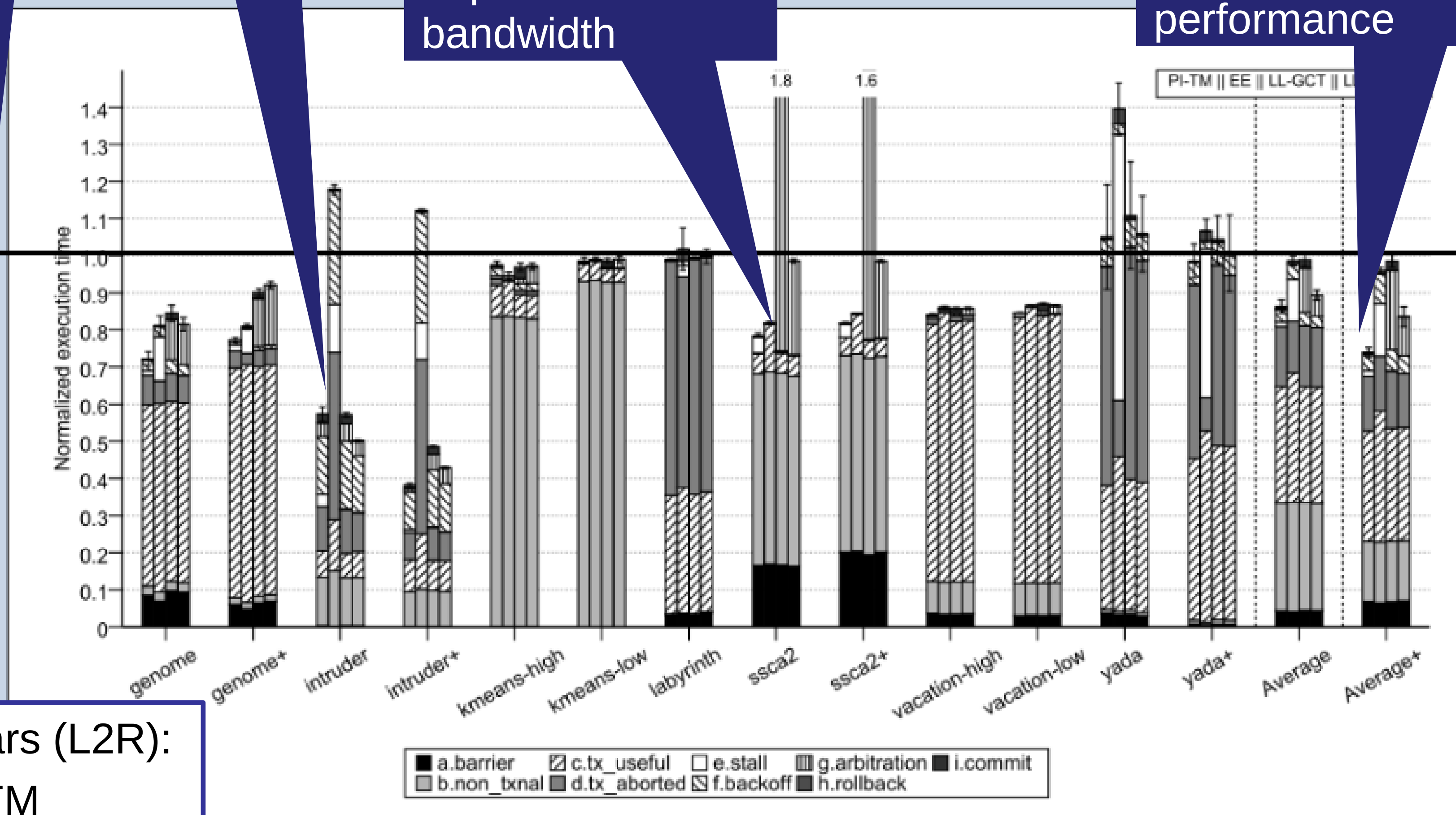
Performance

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