Hardware Transactional Memory in the wild the first year

Konrad Lai
May 2014

The Synchronization Problem

- Writing multi-threaded programs
  - Threads concurrently access shared data
  - Conflicting accesses must be synchronized
- Coarse-Grained Locks
  - Use a *lock* to guard shared data
    - A thread acquires lock, performs updates, and releases lock
    - Other threads wait
  - Downside: serialized all the accesses
- Fine-Grained Locks
  - Assign a lock for each element
    - Accessing different elements need not serialize
  - Such tuning is time consuming and error prone

Transactional Lock Elision

- What we really want
  - Developer effort of coarse-grained locking
  - Performance of fine-grained locking

- **Lock elision**
  - Lock-protected critical sections are *transactionally* executed
  1. Speculatively execute the section without acquiring lock
     - Exposes hidden concurrency
  2. Only enforce serialization when actual data conflicts occur
     - Rollback speculative updates and resume execution
- Main usage model for HTM

Enabling Lock Elision

- Intel TSX: hardware support to enable lock elision
  - Manages transactional updates
    - Transactional reads are tracked, writes are buffered
    - Other threads cannot observe transactional updates
  - Detects data conflicts
    - On actual data conflict, discard updates and rollback registers
    - Restart execution and acquire lock non transactionally
  - Atomically commit modifications
    - Transactional updates are instantaneously visible

Intel TSX Programming Interface

- Two programming interfaces: HLE and RTM
  - HLE: legacy-compatible, policies baked into hardware
  - RTM: more flexible interface, users can fine-tune
- Instruction Set
  - XBEGIN *abort_handler*: start transactional execution
  - XEND: commit transactional execution
  - XTEST/XABORT: test or explicitly abort transactional execution
- Some instructions and events may cause aborts
  - Uncommon instructions (syscalls), interrupts, faults, etc
  - Software must guarantee progress w/ non-transactional path
- RTM ~ ISCA 2013 Herlihy & Moss Hardware TM paper

Commercial HTM

- Commercial Implementations
  - 2011 IBM Blue Gene/Q
  - 2012 IBM zEC12 mainframe
  - 2013 Intel 4th generation Core (Haswell)
  - 2014 IBM POWER8

- Few researchers had experiment with BGQ or zEC
- Haswell is generally in June 2013

Industrial Considerations for HTM

- Provide a clear benefit to customers
  - Improve performance & scalability
  - Ease programmability going forward
- Improve something common and fundamental
  - Widely used critical section/lock-based primitives
- In an easy to use and deploy manner
  - Minimal eco-system impact and effort
  - Clean architectural boundaries
- While managing HW design and validation complexity

HTM Features Convergence

- Convergence over basic functionalities…
  - Best effort HTM
  - Leverage cache coherency protocol/cache(s)
  - Strong Isolation
  - Hardware buffering
  - Reasonable buffer size
  - No instruction count limit
  - Checkpoint of Registers
  - Implicitly Transactional
- Some differences…
  - IBM BGQ supports thread speculation
  - IBM zEC supports constrained transactions
  - IBM POWER8 supports suspend/resume
  - IBM zEC/POWER8 supports non-txn stores (but differently)
  - TX capacity varies medium to large

Intel TSX Case Study: HPC

- SC13
  Performance Evaluation of Intel Transactional Synchronization Extensions for High Performance Computing
  - Richard Yoo, Christopher Hughes, Konrad Lai, Ravi Rajwar (Intel)

Experiment Settings

- Implement lock elision in synchronization library
  - Modified workload macros and pragmas to call our library
  - Optimization: get rid of redundant lock checks
- 4th Generation Core™ Processor with Intel TSX support
  - 4 cores, 2 threads per core
  - L1 data cache (32 KB) buffers transactional updates
  - Conflicts detected at cache line granularity (physical addr)
- Varying range of workloads
  - micros, real-world workloads, user-level TCP/IP stack

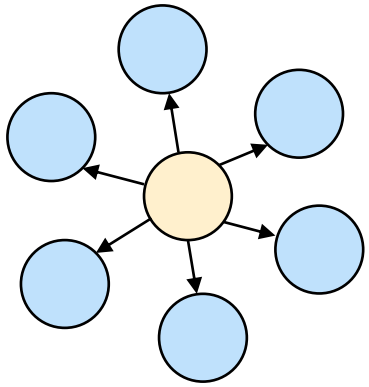
Real-World Workloads

Workload	Description	Sync
graphCluster	Min-cut graph clustering (Kernel 4 of SSCA2)	locks
ua	Unstructured Adaptive (NPB suite)	atomics
physicsSolver	Use PSOR to solve force constraints	locks
nufft	Non-uniform FFT	locks
histogram	Parallel image histogram	atomics
canneal	VLSI router (PARSEC)	lock-free

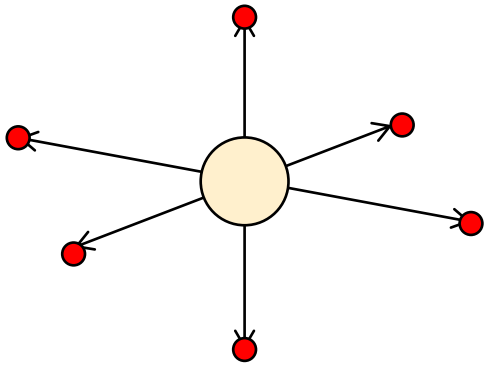
- Computations typically found in the HPC
  - Optimized by domain experts (used to stress Xeon Phi™ processors)
- Apply Intel TSX on top of existing optimizations
  - Locks: simply apply Intel TSX-based lock elision
  - Atomics and lock-free:
    - Convert to regular mem ops, and enclose in lock-protected critical sections
    - Apply Intel TSX-based lock elision

Lockset Elision

- For some workloads, a *set of locks* need to be acquired to enter a critical section
  - **physicsSolver**: acquire two locks for each object pair
  - A pattern frequently observed in HPC



Updating Neighboring Vertices



Particle Represented w/ 6 Points

- Lock acquisition = costly atomic operations
  - Acquiring a set of locks can impose even higher overheads
  - To amortize, substitute w/ single transactional begin

Transactional Coarsening

- Batch transactions to better amortize overhead
  - *Static* coarsening: merge different critical sections into one
  - *Dynamic* coarsening: combine multiple dynamic instances

```
#pragma omp atomic
    tmor[ig1] += tx[il1] * third;
#pragma omp atomic
    tmor[ig2] += tx[il2] * third;
#pragma omp atomic
    tmor[ig3] += tx[il3] * third;
#pragma omp atomic
    tmor[ig4] += tx[il4] * third;
```

ua Code Example

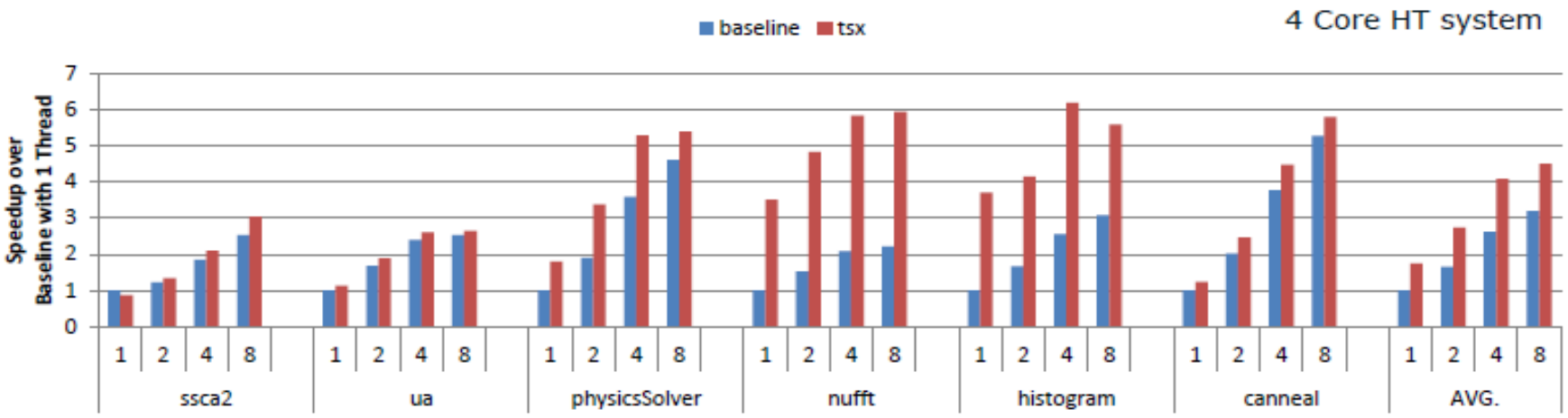
```
for (int y = y0; y < y1; y++)
    for (int x = x0; x < x1; x++) {
        if (x % TXNGRAN == 0)
            TM BEGIN();
        // Update histogram bin
        UPDATE BIN(x);
        if (x % TXNGRAN == TXNGRAN - 1)
            TM_END();
    }
```

histogram Code Example

- Tradeoff:
  - Larger transaction = lower overheads, but higher conflict probability
  - Automated tuning approach could be implemented

Code example for illustration purpose only

TSX Evaluation on HPC Workloads



Substitute atomic operations, locks, and non-blocking sync. with RTM

- Average 1.41x speedup with 8 threads

Workloads benefit from RTM by

1. Exploiting concurrency within a critical section (**nufft**)
2. Reducing the synchronization cost (**ssca2, physicsSolver, nufft, histogram**)
3. Replacing complex non-blocking sync. w/ regular memory ops (**canneal**)

Other Results

- RMS-TM
  - Memory allocation and system call within transactional regions
  - Intel TSX provides similar performance to fine-grained lock
- User-Level TCP/IP Stack
  - Modify user-level TCP/IP stack in PARSEC
  - Average 1.31x bandwidth improvement
  - Need conditional variable TM optimization
- Java + Intel TSX
  - Transactionally execute `synchronized` statements
  - Promising initial results

PARSEC 3 User-Level TCP/IP Stack

- Conditional variable is common in real apps
- Cond_wait always abort
- Busy wait instead
- 1.3x better BW

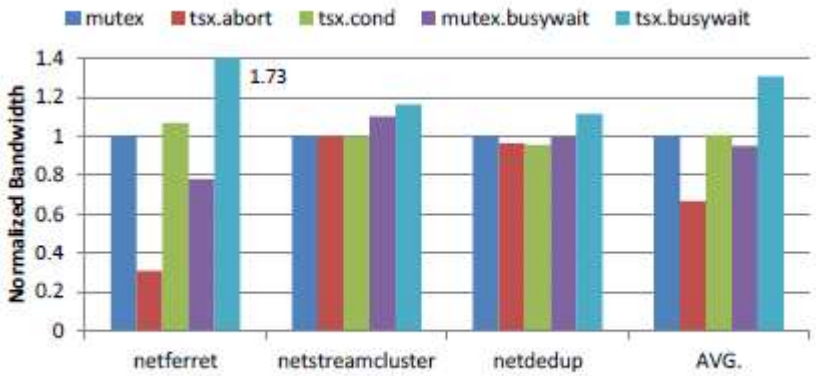


Figure 6: Intel TSX performance on user-level TCP/IP stack. Reports server-side read bandwidth.

```
pthread_mutex_lock(&lock);

while (monitor state not true) {
    // Wait till condition met
    pthread_cond_wait(&lock, &cond);
}

pthread_mutex_unlock(&lock);
```

Listing 4: PThread condition variable wait routine.

```
pthread_mutex_lock(&lock);

... update monitor state to true ...

// Signal waiting thread
pthread_cond_signal(&cond);

pthread_mutex_unlock(&lock);
```

Listing 5: PThread condition variable signal routine.

```
pthread_mutex_lock(&lock);

while (monitor state not true) {
    // Busy-wait till condition met
    pthread_mutex_unlock(&lock);
    pthread_mutex_lock(&lock);
}

pthread_mutex_unlock(&lock);
```

Listing 6: Busy-wait substitution for conditional wait.

Intel TSX Case Studies: Databases

- HPCA 2014
  Improving In-Memory Database Index Performance with Intel® Transactional Synchronization Extensions
  - Tomas Karnagel (TU Dresden), Roman Dementiev (Intel), Ravi Rajwar (Intel), Konrad Lai (Intel), Thomas Legler (SAP AG), Benjamin Schlegel (TU Dresden), Wolfgang Lehner (TU Dresden)

Challenge: Implementing Concurrency Control

- High-performance in-memory databases
  - Extensively use low-level synchronization mechanisms
  - Synchronize access to internal in-memory data structures
- Rich history w/ numerous proposals, implementations
  - Remains complicated and difficult to verify/deploy/productize
    - Trade-off – complexity/verifiability
    - Latches/Locks remain popular
  - Consumes significant developmental effort and resources

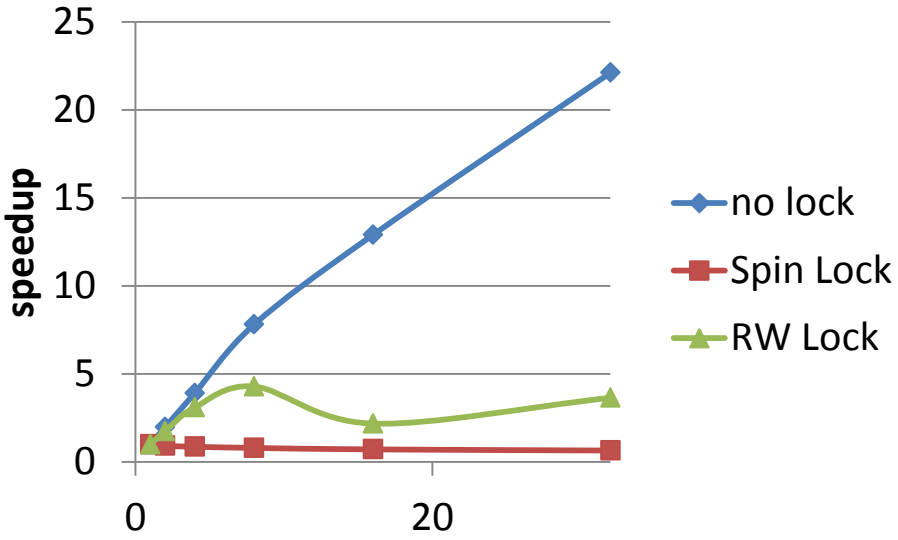
"Perhaps the most urgently needed future direction is simplification. Functionality and code for concurrency control and recovery are too complex to design, implement, test, debug, tune, explain, and maintain."

- Graefe, 2010

A Case Study: Two Index Implementations

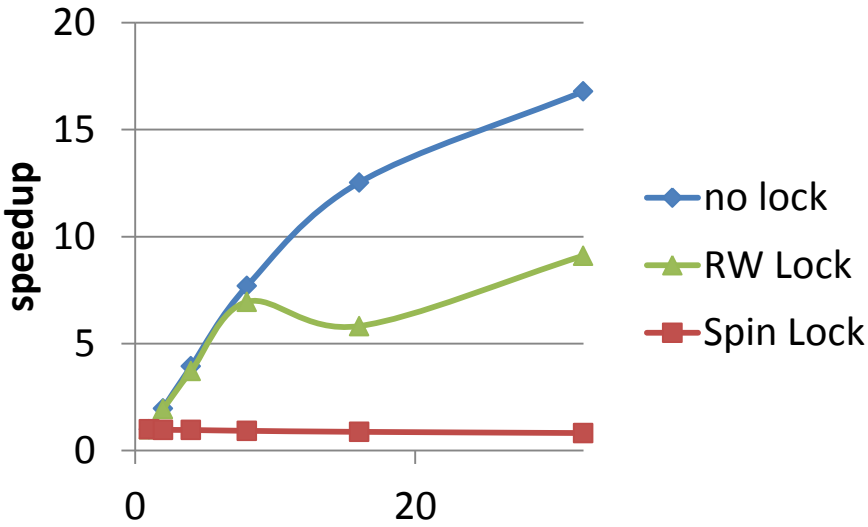
B+Tree Index

(a common index implementation)



Delta Storage Index

(from the SAP HANA® database)

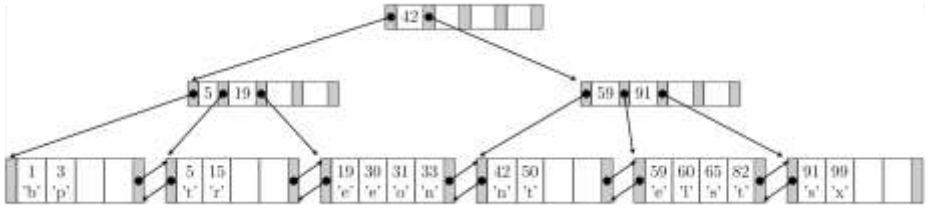


- ***Read-Only*** Queries on Dual Socket Intel® Xeon® E5-2680 Server

Hidden Scalability Impact of Atomic Read-Modify-Write Operations

Case Study: Index Tree Implementations

- SAP HANA Database
  - Read optimized column store database system
- Two index implementations
  - B+Tree [Data Structure]
    - Common implementation
    - Smaller foot print

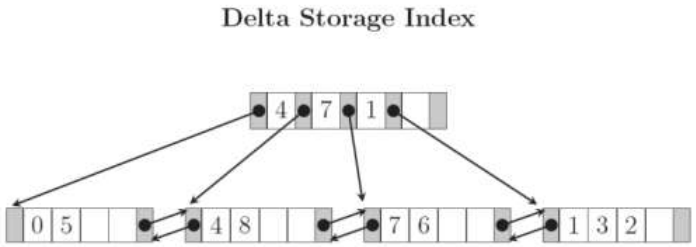


  - Delta Storage Index (B+Tree with a Dictionary)
    - Complex data structure with additional structures
    - Large foot print

Lock protect access
  - Reader-Writer
  - Spin Lock

Column Data
1
7
2
3
1
0
8
2
...

Dictionary	
ID	Entry
0	Bakersfield
1	San Francisco
2	Santa Clara
3	San Jose
4	Fresno
5	Berkeley
6	Sacramento
7	Palo Alto
8	Oakland



Steps To Apply Intel TSX

- Modify synchronization library for lock elision
  - Use either the HLE or RTM interface
  - Application changes not immediately required
- RTM: Start with good “fallback handler" design
  - Avoid “the lemming effect”
  - Retry appropriately for both conflict and capacity aborts
- Analyze early results
  - Extensive hardware profiling infrastructure
- Apply transaction-friendly transformations if needed
  - To reduce conflict and capacity aborts

Lemming Effect

```
XA : xbegin; test; xabort; (retry loop when lock is busy)
L—U: Lock; critical section; Unlock (non-transactional execution)
T1 --AL-------------UXAXAXAXAXAssssssssssssssssL----------UXAXA
T2 ---AXAXAXAXAXAsssL-------------UXAXAXAXAXAXAssssssssssssL---
T3 ---AXAXAXAXAXAssssssssssssssssL----------UXAXAXAXAXAssssss
```

Persistent convoy of non-transactional execution

- Happens when a thread aborts and acquires the lock
  - And aborts other threads
- And other threads immediately retry transactional execution
  - Find the lock held and abort
    - And immediately retry transactional execution
  - Reach their retry limit
    - And enter non-transactional fallback execution, trying to acquire lock
  - And never actually had any opportunity to elide the lock

Elision is effectively disabled until all threads have serially released the lock

- Disabled forever if at least 1 thread is holding the lock

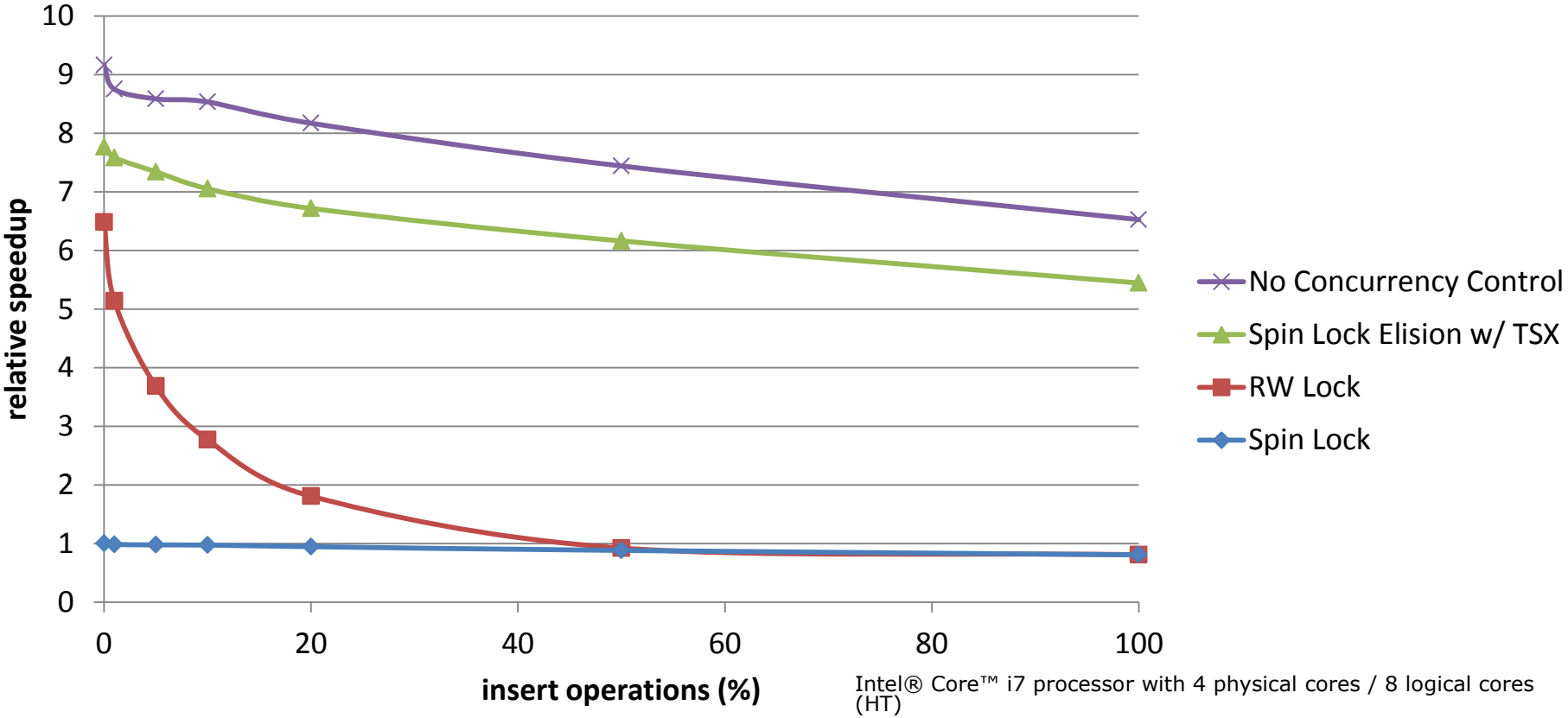
Lemming Effect

- Fix is simple
  - Don’t retry until the lock is free (test-and-test-&-set)

- `T1 --AL-------------UX-------`
- `T2 ---Asssssssssssss X-------`
- `T3 ---Asssssssssssss X-------`

Initial Results: B+Tree

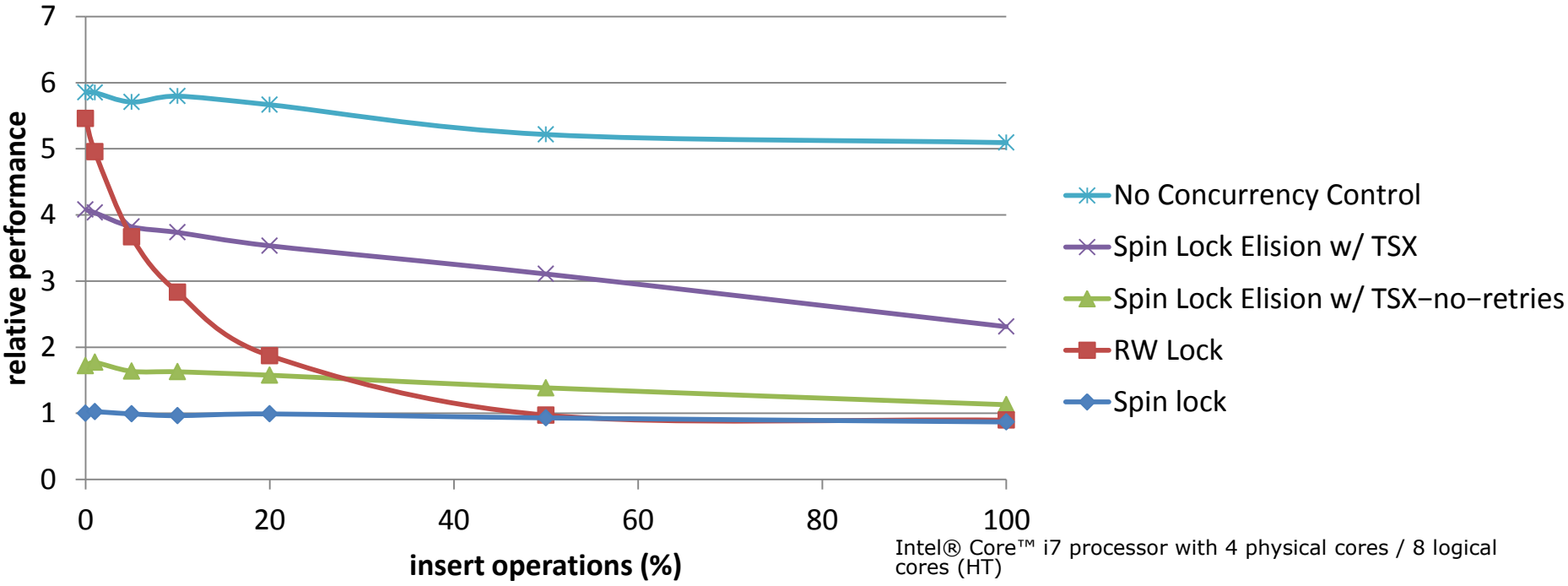
- Intel TSX provides significant gains with no application changes
  - Outperforms RW lock on read-only queries
  - Significant gains with increasing inserts (6x for 50%)



Intel® Core™ i7 processor with 4 physical cores / 8 logical cores (HT)

Initial Results: Delta Storage Index

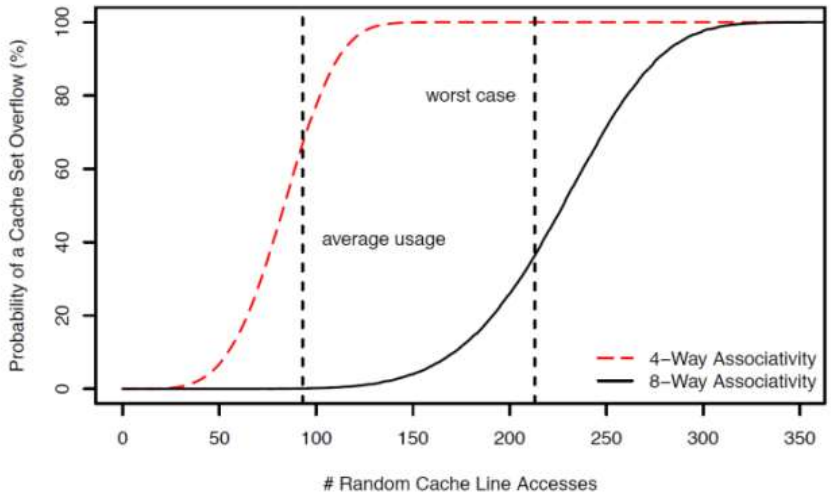
- Intel TSX provides gains with no application changes
  - Different profile as compared to B+Tree
  - Spin lock w/ Intel TSX better than RW Lock when > 5% insert
    - Significant gap as compared to no concurrency control
- Baseline should implement good retry policy on aborts



Intel® Core™ i7 processor with 4 physical cores / 8 logical cores (HT)

Analysis: Delta Storage Index

- Capacity Aborts
  - Cache Associativity Limits
    - Aborts typically before cache size limits
    - Hyper-threads share the L1 cache
  - Dictionary contributes to larger footprint
  - Algorithmic level
    - Node/Leaf Search Scan
    - O(n) random lookups
- Data Conflicts
  - Single dictionary
  - Global memory allocator



Well Known Causes of Transactional Aborts

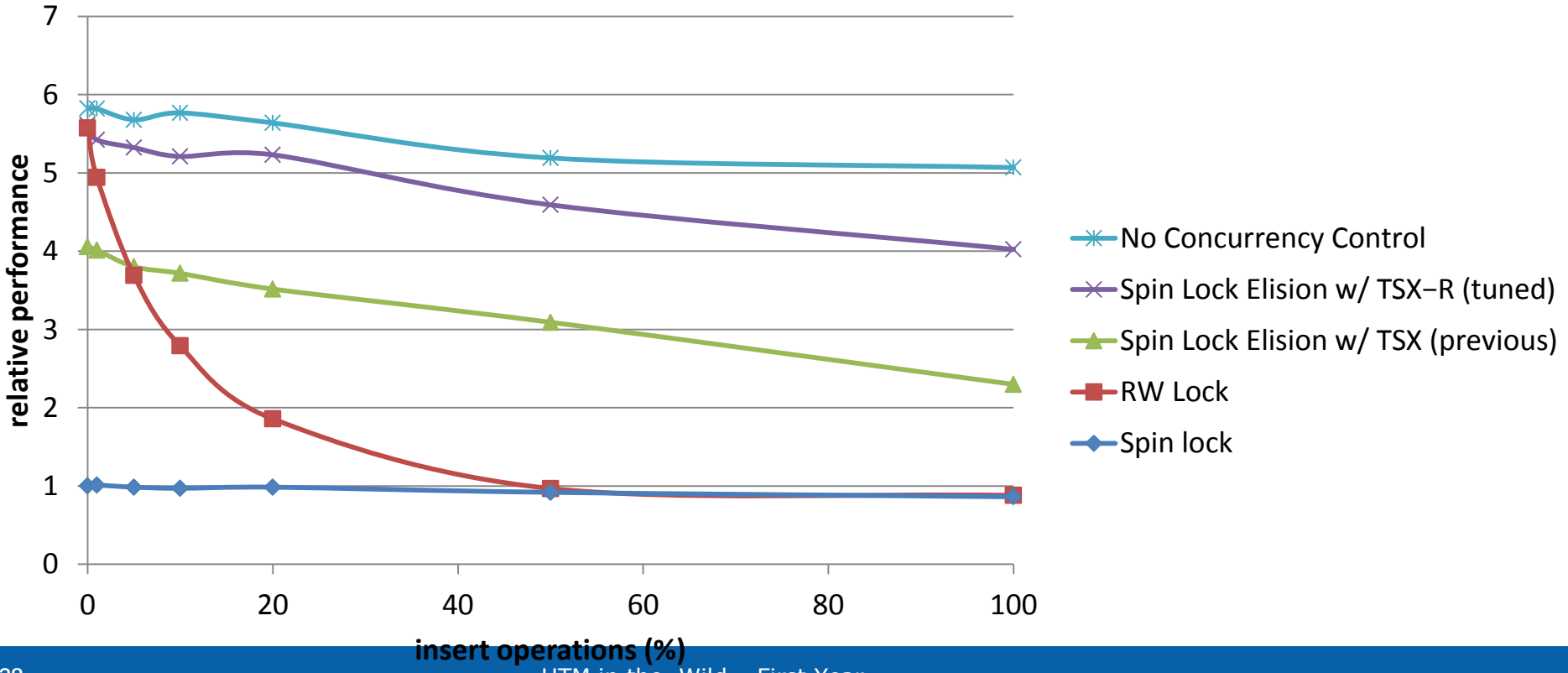
Software Transformations

- Capacity Aborts
  - Node/Leaf Search Scan
    - Causes O(n) random lookups
  - Transformation – Binary Search
    - Causes O(log(n)) random lookups
- Data Conflicts
  - Single dictionary
  - Global memory allocator
  - Transformation – Multiple Dictionaries, per-thread/core allocators

Well Known Transformations

Tuned Results: Delta Storage Index

- Intel TSX provides significant gains with transformations
  - Restores read-only query performance
  - Spin lock w/ Intel TSX significantly outperforms RW lock (5x for 50% inserts)
  - Close to 'No Concurrency Control"



Lessons

- Lemming Effect
  - A common but deadly mistake
  - Simple fixes
- Capacity Limits
  - Best determined by running actual workloads
  - Transient aborts due to associativity
- Use Analysis Tools and Iterate Optimizations
  - Extensive infrastructure
  - Can pinpoint areas of interest effectively
- Retry on Aborts
  - Limited retries are effective – even on capacity aborts
  - Abort rates can be misleading

Intel TSX Case Studies: Java

- TRANSACT 2014.
  Early Experience on Transactional Execution of Java Programs Using Intel Transactional Synchronization Extensions
  - Richard Yoo, Sandhya Viswanathan, Vivek Deshpande, Christopher Hughes and Shirish Aundhe (Intel)

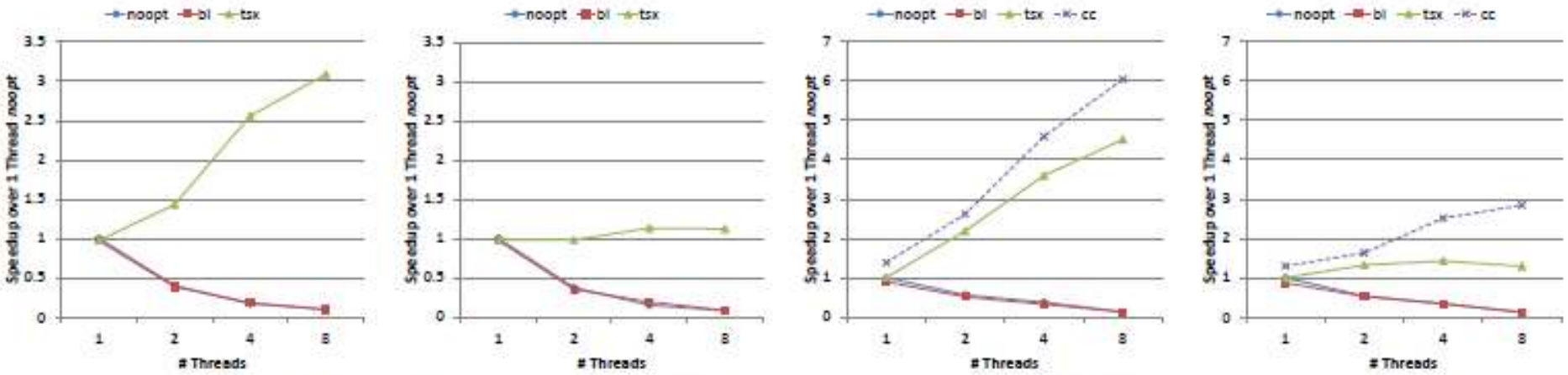
HTM to elide fat lock

- Java Sync Block use thin/fat lock
  - Thin – uncontended
  - Fat – contended
- Adaptive (see paper)
- @JIT/JVM no src changes
  - Tuning needed

```
routine monitor_enter(monitor_lock) {

    // Handle thin lock
    if acquire_thin_lock(monitor_lock) is success,
        return;

    // Lock is inflated: try to elide fat lock
    for (int i = 0; i < retry_threshold; i++) {
        // Increment total # txns tried
        inc_total_count(monitor_lock);

        XBEGIN abort_handler;

        // Check if fat lock is already acquired
        if not fat_lock_occupied(monitor_lock),
            // Proceed to execute the critical section
            return;

        else
            // Fat lock is acquired by another thread
            XABORT;

abort_handler:
        adjust_elision(monitor_lock);
    }

    // Elision failed: explicitly acquire fat lock
    acquire_fat_lock(monitor_lock);
}
```

Listing 3: **Pseudocode for Intel TSX-enabled monitor lock acquisition.**

Hashtable & HashMap



(a) `Hashtable` **(0p, 100g, 0r)** (b) `Hashtable` **(20p, 70g, 10r)** (c) `HashMap` **(0p, 100g, 0r)** (d) `HashMap` **(20p, 70g, 10r)**

Figure 1: **Microbenchmark performance results. Speedup is against** *noopt* **with single thread.** *p*, *g*, **and** *r* **denotes the percentage of** *put()*, *get()*, **and** *remove()* **operations, respectively.**

TSX usages in Database

- ICDE 2014
  Exploiting Hardware Transactional Memory in Main-Memory Databases
  - Viktor Leis, Alfons Kemper, Thomas Neumann (TU Munchen)
- EuroSys 2014
  Using Restricted Transactional Memory to Build a Scalable In-Memory Database.
  - Zhaoguo Wang, (Fudan University & Shanghai Jiao Tong University), Hao Qian (Shanghai Jiao Tong University), Jinyang Li (New York University), Haibo Chen (Shanghai Jiao Tong University)

HTM to implement DB Transaction

- Classical DB OCC (Optimistic Concurrency Control)
- HTM can’t handle DB transactions directly
  - Too small
  - Hybrid approach not worth it
- HTM uses in put() and get() with versioning
- HTM uses in validate() and commit()
  - Take advantages of larger HTM readset than writeset

Emerging TSX Usages

- Ruby GIL Elision (IBM)
- Race Detection
- Virtual machine Introspection
- GC
- …

Some Observations

- Hello World of HTM
  - Atomic { i++; }
- Common Implementation questions:
  - Size
  - But do you know your need? Not quite
- Common Functional Bugs
  - No fall back code, or retry forever
  - Fall back code and HTM code doesn't work together
- Common Performance Bugs
  - Lemming Effect. Sustained non-HTM execution.
  - Central malloc (always conflicts). Use tcmalloc or like
  - False sharing.
  - Uncritical stats.
- TM Programming Model still Work-in-Progress
  - No medium size apps in the last 5 years

Time to Experiment

- Time to do now
- Play with it
- Experiment
- HW is settled for now

Background

TSX References

- Intel TSX Specifications
  - *"Intel architecture instruction set extensions programming reference."* Chapter 8: Intel transactional synchronization extensions
- Enabling and Optimization Guidelines
  - *"Intel 64 and IA-32 architectures optimization reference manual."* Chapter 12: Intel TSX recommendations
- Additional Resources for Intel TSX
  - http://www.intel.com/software/tsx

Papers using Intel TSX

IPDPS 2014 (5/2014)

- Performance and Energy Analysis of the Restricted Transactional Memory Implementation on Haswell - Bhavishya Goel (Chalmers University of Technology); Rubén Titos (University of Murcia); Anurag Negi (Chalmers University Of Technology); Sally A. McKee (Chalmers University of Technology); Per Stenstrom (Chalmers University of Technology)

EuroSys 2014 (4/2014)

- StackTrack: An Automated Transactional Approach to Concurrent Memory Reclamation - Dan Alistarh (MIT), Patrick Eugster (Purdue University), Maurice Herlihy (Brown University), Alexander Matveev (Tel-Aviv University), Nir Shavit (MIT and Tel-Aviv University)
- Using Restricted Transactional Memory to Build a Scalable In-Memory Database - Zhaoguo Wang, (Fudan University & Shanghai Jiao Tong University), Hao Qian (Shanghai Jiao Tong University), Jinyang Li (New York University), Haibo Chen (Shanghai Jiao Tong University)
- Algorithmic Improvements for Fast Concurrent Cuckoo Hashing - Xiaozhou Li (Princeton University), David G. Andersen (Carnegie Mellon University), Michael Kaminsky (Intel Labs), Michael J. Freedman (Princeton University)

Euro-TM 2014 (4/2014)

- Optimized Single Global Lock Fallback - Irina Calciu, Justin Gottschlich, Tatiana Shpeisman, Gilles Pokam and Maurice Herlihy
- Workload-Oblivious Self-Tuning of Intel TSX - Nuno Diegues and Paolo Romano

ICDE 2014 (4/2014)

- Exploiting Hardware Transactional Memory in Main-Memory Databases - Viktor Leis, Alfons Kemper, Thomas Neumann (Technische Universität München, Germany)

WoDet 2014 (3/2014)

- Accelerating Precise Race Detection Using Commercially-Available Hardware Transactional Memory Support - Hassan Salehe Matar, Ismail Kuru, Serdar Taşıran (Koç University), Roman Dementiev (Intel)

Papers using Intel TSX (cont)

TRANSACT 2014 (3/2014)

- Improved Single Global Lock Fallback for Best-effort Hardware Transactional Memory (Irina Calciu, Tatiana Shpeisman, Gilles Pokam and Maurice Herlihy)
- Early Experience on Transactional Execution of Java Programs Using Intel Transactional Synchronization Extensions (Richard Yoo, Sandhya Viswanathan, Vivek Deshpande, Christopher Hughes and Shirish Aundhe)
- STAMP Need Not Be Considered Harmful (Wenjia Ruan, Yujie Liu and Michael Spear)
- Reduced Hardware NOREC: An Opaque Obstruction-Free and Privatizing HyTM (Alex Matveev and Nir Shavit)
- Exploring the Performance and Programmability Design Space of Hardware Transactional Memory (Mike Dai Wang, Mihai Burcea, Linghan Li, Sahel Sharifymoghaddam, Greg Steffan and Cristiana Amza)
- Improve HTM Scaling with Consistency-Oblivious Programming (Hillel Avni and Bradley Kuszmaul)
- A Hybrid TM for Haskell (Ryan Yates and Michael Scott)
- Transaction-Friendly Condition Variables (Chao Wang, Yujie Liu and Michael Spear)

HPCA 2014 (2/2014)

- Concurrent and Consistent Virtual Machine Introspection with Hardware Transactional Memory - Yutao Liu (Shanghai Jiao Tong University), Yubin Xia (Shanghai Jiao Tong University), Haibing Guan (Shanghai Jiao Tong University), Binyu Zang (Shanghai Jiao Tong University), Haibo Chen (Shanghai Jiao Tong University)
- Improving In-Memory Database Index Performance with Intel® Transactional Synchronization Extensions - Tomas Karnagel (TU Dresden), Roman Dementiev (Intel), Ravi Rajwar (Intel), Konrad Lai (Intel), Thomas Legler (SAP AG), Benjamin Schlegel (TU Dresden), Wolfgang Lehner (TU Dresden)

PPoPP 2014 (2/2014)

- Eliminating Global Interpreter Locks in Ruby through Hardware Transactional Memory - Rei Odaira, Jose Castanos and Hisanobu Tomari

Papers using Intel TSX (cont)

DMTM 2014 (1/2014)

- The moment of truth: are we done with STM? - Nuno Diegues, Paolo Romano and Luís Rodrigues
- Enhancing Real-Time Behaviour of Parallel Applications using Intel TSX - Florian Haas, Stefan Metzlaff, Sebastian Weis and Theo Ungerer

SC13 (11/2013)

- Performance Evaluation of Intel Transactional Synchronization Extensions for High Performance Computing - Richard Yoo, Christopher Hughes, Konrad Lai, Ravi Rajwar

HPTS 2013 (9/2013)

- Hardware Transactional Memory on Haswell - Viktor Leis (TU Munchen)

ApSys 2013 (7/2013)

- Opportunities and pitfalls of multi-core scaling using Hardware Transaction Memory - Zhaoguo Wang, Hao Qian, Haibo Chen & Jinyang Li