

# Hardware Transactional Memory on Haswell

Viktor Leis

Technische Universität München



# Introduction

- ▶ transactional memory is a very elegant programming model

```
transaction {  
   $a = a - 10;$   
   $b = b + 10;$   
}  
Transaction 1
```

```
transaction {  
   $c = c - 20;$   
   $a = a + 20;$   
}  
Transaction 2
```

# Transactional Memory Implementations

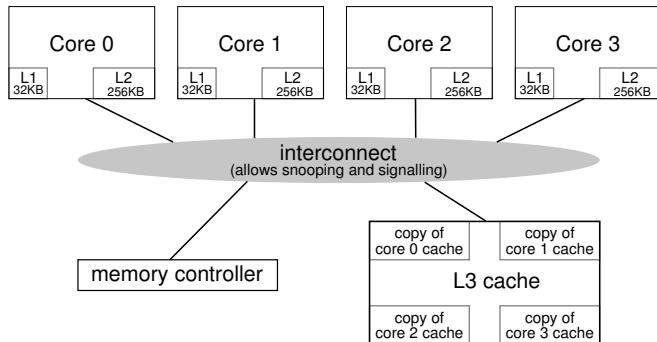
- ▶ idea: keep track of read/write sets, abort on read/write or write/write conflict
- ▶ software transactional memory (hash table)
- ▶ Sun Rock: store queue (32 entries)
- ▶ IBM Blue Gene/Q: multi-versioned L2 cache (20MB)
- ▶ Intel Haswell: L1 cache (32KB)

# Intel Transactional Synchronization Extensions

- ▶ Restricted Transactional Memory (RTM):
  - ▶ XBEGIN: begin
  - ▶ XEND: commit
  - ▶ XABORT: rollback
- ▶ Hardware Lock Elision (HLE):
  - ▶ XACQUIRE prefix: acquire lock speculatively
  - ▶ XRELEASE prefix: release lock speculatively

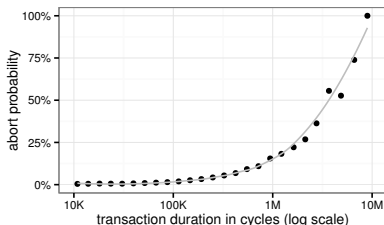
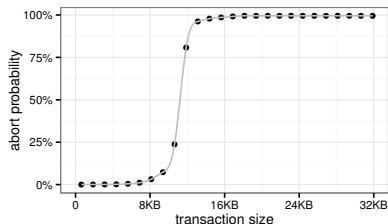
# Cache Coherency

- ▶ cache coherency protocol is used to detect conflicts
- ▶ L1 cache serves as a buffer
- ▶ tracking is done at cache line granularity (64 bytes)



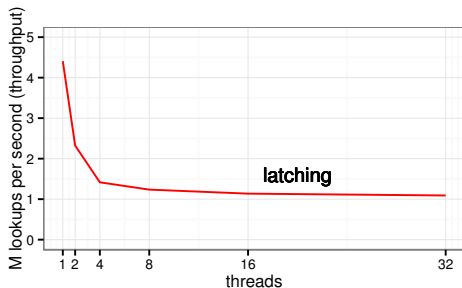
# Limitations of Haswell's HTM

- ▶ size (32KB) and associativity (8-way) of L1 cache limit transaction size
- ▶ interrupts, context switches limit transaction duration
- ▶ certain (fairly uncommon) instructions always cause abort
- ▶ no forward-progress guarantees (fallback to lock necessary)



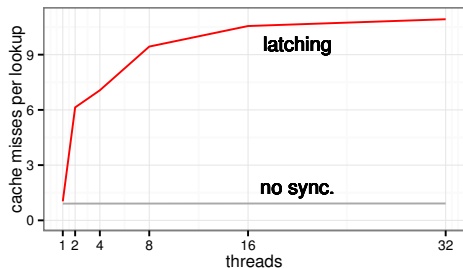
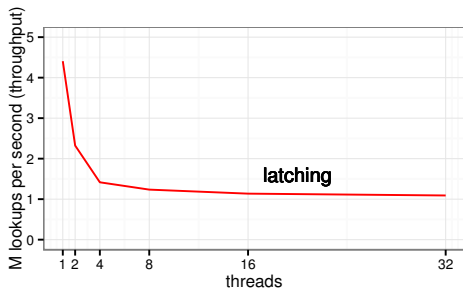
## The Problem with Locks

- ▶ 4-socket, 32-core Nehalem EX
- ▶ lookups in an Adaptive Radix Tree (16M dense integer keys)
- ▶ fine-grained read-write spinlocks
- ▶ no logical contention
- ▶ lock acquisition causes write
- ▶ cache coherency misses destroy performance (“cache line ping pong”)



# The Problem with Locks

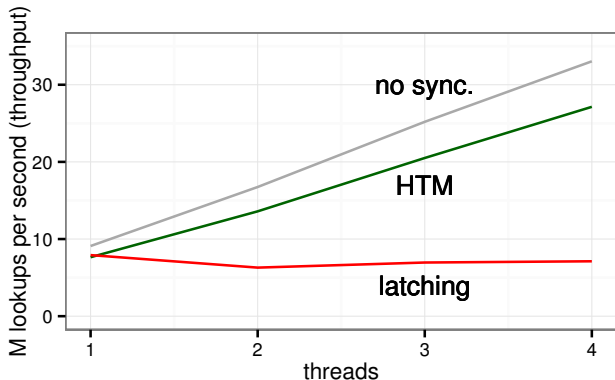
- ▶ 4-socket, 32-core Nehalem EX
- ▶ lookups in an Adaptive Radix Tree (16M dense integer keys)
- ▶ fine-grained read-write spinlocks
- ▶ no logical contention
- ▶ lock acquisition causes write
- ▶ cache coherency misses destroy performance (“cache line ping pong”)





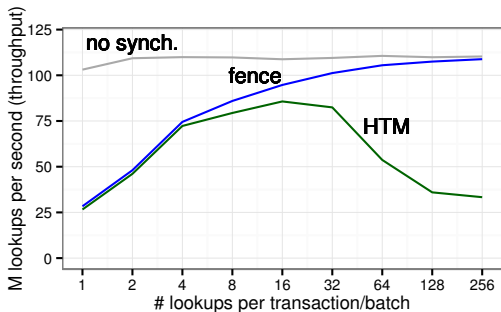
# HTM Performance

- ▶ 4-core Haswell
- ▶ lookups in an Adaptive Radix Tree (16M dense integer keys)
- ▶ lookups depend on each other
- ▶ no cache coherency misses with HTM



# Transaction Granularity

- ▶ out-of-order execution complicates the picture
- ▶ independent lookups
- ▶ small transactions: memory fencing overhead
- ▶ large transactions: capacity limitations



# Lock-Free Data Structures

- ▶ avoid the problem of cache coherency misses
- ▶ without HTM the lock-free synchronization (e.g., Hekaton) is necessary for good performance
- ▶ lock-free data structures are complex
- ▶ and often require additional indirections (e.g., delta records and page table for BW-tree)

# Database Transactions

- ▶ HTM can replace fine-grained latching (without complex lock-free data structures)
- ▶ but we want ACID and transactions of arbitrary size
- ▶ idea: use HTM as a building block

► [www.hyper-db.com](http://www.hyper-db.com)

The screenshot shows the HyPer website homepage. The main heading is "HyPer" followed by "A Hybrid OLTP&OLAP High-Performance Database System". It describes HyPer as a hybrid online transactional processing (OLTP) and online analytical processing (OLAP) high-performance main memory database system. Key performance metrics are highlighted: OLTP (> 100,000 single-threaded TPC-C TX/s on modern commodity hardware) and OLAP (best-of-breed response times), operating simultaneously on the same database. Navigation buttons for "Learn more" and "Try it out" are present.

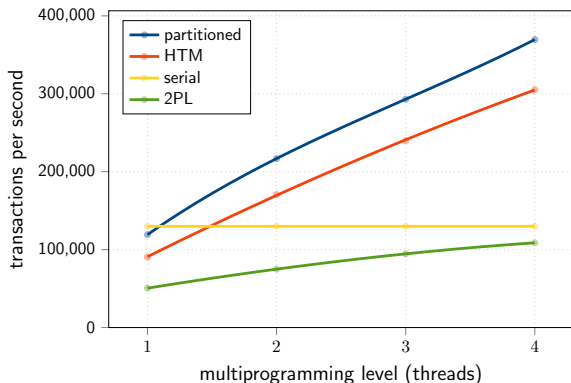
The "Highlights" section includes:

- In-memory Data Management:** HyPer relies on in-memory data management without the overhead of traditional database systems. SQL table definitions are transformed into simple vector-based virtual memory representations.
- Efficient Snapshotting:** OLAP query processing is separated from OLTP transaction processing by forking virtual memory snapshots.
- Data-centric Code Generation:** Transactions and queries are specified in SQL or a PL/SQL-like scripting language and are efficiently compiled into efficient LLVM assembly code.
- No compromises:** HyPer's transaction processing is fully ACID-compliant. Queries are specified in SQL-92 plus some extensions from subsequent standards.

The diagram illustrates the "Virtual Memory Management" architecture. It shows a grid of "Pageable" memory blocks. An "OLTP Requests/Tx" arrow points to a specific block containing data 'a', 'b', 'c', 'd', 'e'. An "Update a' -> a'" arrow points to a block containing 'a'', 'b', 'c', 'd', 'e'. An "OLAP-Session" arrow points to a block containing 'a', 'b', 'c', 'd', 'e'. A "Read a" arrow points to a block containing 'a', 'b', 'c', 'd', 'e'. A "Copy on write" arrow points from the OLTP block to the OLAP block. The diagram also shows a "Pageable" column on the right side of the grid.

# HTM-based Concurrency Control

- ▶ split database transaction into multiple HTM transactions
- ▶ “glue” together HTM transactions with timestamp ordering
- ▶ TPC-C with 32 warehouses



# Conclusions

- ▶ HTM is a new synchronization primitive that can improve performance and simplify the implementation at the same time
- ▶ very good fit with high-performance database systems

## References

- ▶ “Exploiting Hardware Transactional Memory in Main-Memory Databases”, Viktor Leis, Alfons Kemper, Thomas Neumann (forthcoming)
- ▶ Intel Developer Forum 2012 presentation:  
<http://software.intel.com/sites/default/files/blog/393551/sf12-arcs004-100.pdf>
- ▶ Instruction Set Reference (chapter 8):  
<http://download-software.intel.com/sites/default/files/319433-014.pdf>
- ▶ TSX recommendations (chapter 12):  
<https://www-ssl.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf>
- ▶ David Kanter’s analysis:  
<http://www.realworldtech.com/haswell-tm/>