Fachhochschule Münster University of Applied Sciences

Evaluation of Transactional Memory and Other Techniques to Improve the Performance of Algorithms in High Energy Physics for New Processor Architectures

Philipp Schoppe

CERN PH-SFT / University of Applied Sciences Münster

ROOT Team Meeting November 28, 2014

Agenda

Traditional Concurrency Control
- Introduction
- Mutual Exclusion
- Mutex Drawbacks
- Lock-free Data Structures
- Painful State of the Art

Transactional Memory
- Introduction
- Major Benefits
- Status
- Performance

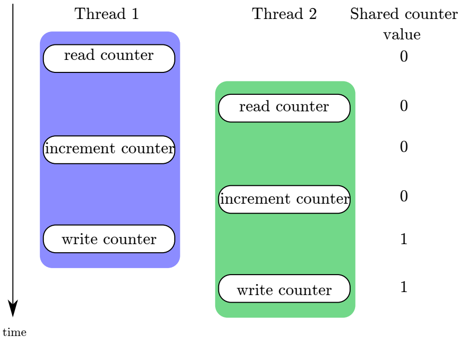
Conclusion and Outlook

Traditional Concurrency Control

Traditional Concurrency Control

Introduction

- Managing shared resources is critical
- Ensure ordered access to shared data
- Atomic hardware instructions
  - `Test-and-set`
  - `atomic-increment`
  - `CAS`
  - `LL/SC`
- Memory barriers
  - acquire barrier
  - release barrier
  - full barrier



Traditional Concurrency Control

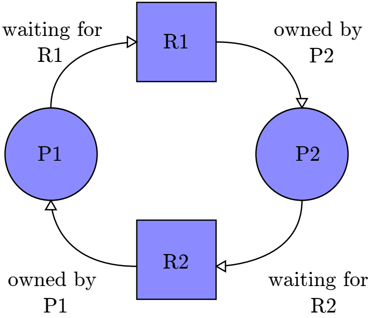
Mutual Exclusion

- Avoid data races
- Critical section executed by one thread at a time
- Serialise access to shared data
- Locking
  - Mutex
  - Spinlock
  - Readers-Writer lock

Traditional Concurrency Control

Mutex Drawbacks

- **Deadlock**
  - Processes lock a set of objects with two or more mutexes and they each wait for the lock owned by another thread.
- **Priority inversion**
  - A low priority process may hold a lock that is needed by a high priority process
- **Convoying**
  - A process may be descheduled or interrupted while holding a lock.



Traditional Concurrency Control

Lock-free Data Structures

- Mutual exclusion is based on blocking an active process, if necessary
  $\Rightarrow$ Lock-free and wait-free data structures

Maurice Herlihy:

Definition (Lock-free)

A concurrent data structure is **lock-free**, if a process is guaranteed to complete an operation on it after the system as a whole takes a finite number of steps.

Traditional Concurrency Control

Lock-free Data Structures

- Mutual exclusion is based on blocking an active process, if necessary
  $\Rightarrow$ Lock-free and wait-free data structures

Maurice Herlihy:

Definition (Lock-free)

A concurrent data structure is **lock-free**, if a process is guaranteed to complete an operation on it after the system as a whole takes a finite number of steps.

Definition (Wait-free)

A concurrent data structure is **wait-free**, if each process is guaranteed to complete an operation on it after taking a finite number of steps.

Traditional Concurrency Control

Lock-free Data Structures

- Lock-freedom has been subject to research for years
- Only few efficient and correct implementations to a very limited range of data structures are known
- A working algorithm is almost always a publishable result
- Wait-freedom with good performance is even harder to achieve
- **Extremely** complex to implement!
  - Herb Sutter talks:
    Atomic<> Weapons: The C++ Memory Model and Modern Hardware ▸ Video
    Lock-Free Programming (or, Juggling Razor Blades) ▸ Video

Traditional Concurrency Control

Painful State of the Art

- Joe Duffy: Solving 11 Likely Problems In Your Multithreaded Code ▸ Article
  - Forgotten Synchronization
  - Incorrect granularity
  - Read and write tearing
  - Lock-free reordering
  - Lock convoys
  - Priority inversion
  - Incomposability
  - ...

Lockfree-reordering

```
Initialization:
    bool gPrintFlag  = false;
    int  gPrintValue = 0;

Thread 1:
    gPrintValue = 2014;
    gPrintFlag  = true;

Thread 2:
    while (gPrintFlag == false) { }

    std::cout << gPrintValue;
```

Transactional Memory

Transactional Memory

Introduction

- "Transactional Memory: Architectural Support for Lock-Free Data Structures" Paper
- Database-style transactions working on shared memory
- ACI(D)
  - **Atomicity:** either all operations take effect, or nothing happens
  - **Consistency:** a transaction can only commit legal results, leaving the system in a valid state
  - **Isolation:** operations within a transaction are hidden from other, concurrently running transactions
  - **Durability:** when successfully committing, a transaction's changes are guaranteed to be permanent
- Optimistic speculation
- Extension to the cache-coherence protocol

Transactional Memory

Major Benefits

- Makes lock-free synchronization easily accessible
- Composability
  - "Generic Programming Needs Transactional Memory" ▸ Paper
- Easy to use

Transactional block

```
int shared_data[20];

int
set_shared_data(int index, int value)
{
    __transaction_atomic {
        shared_data[index] = value;
    }
}
```

Transactional Memory

Status

- Many Software Transactional Memory (STM) libraries available
- Intel released Transactional Synchronization Extensions (TSX) in the end of 2013
  - But it contains a bug ... PDF
- Velox stack Overview
  - Applications
  - Benchmarks
  - Compilers
  - Libraries, system libraries
  - Kernel scheduler
- Ongoing integration effort into the C++ standard

Transactional Memory

Performance

- STM deemed inefficient
- Performance is often not compared to traditional synchronization in literature
- Hardware TM as a solution?
- Evaluation of TM during my master thesis ▸ PDF
  - Experimental evaluation for queue and simple histogram
  - Results from other literature and research

Transactional Memory

Benchmark System

- Intel Core i7-4790, quad core CPU with eight threads
  - Each core runs at 3.60 GHz
  - 32 KB of L1 data cache
  - 64 bytes cache line size
  - 16 GB RAM

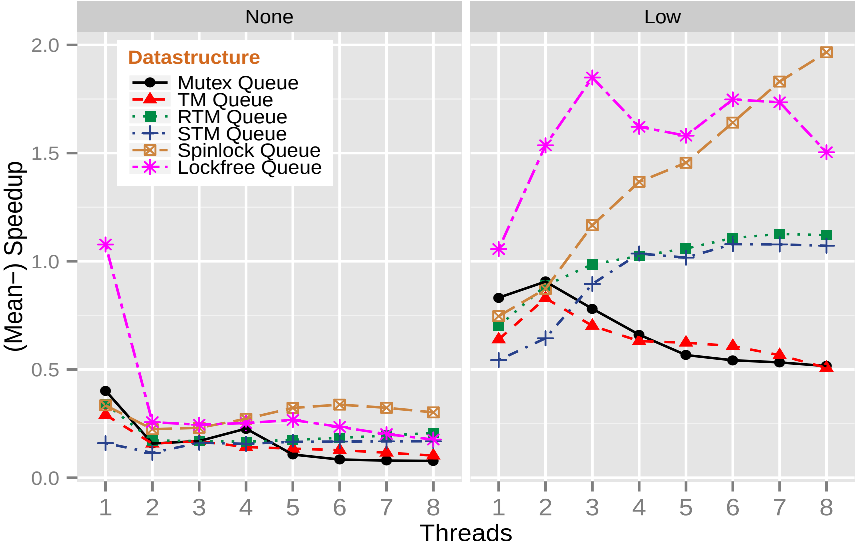
Transactional Memory

Benchmark Setup

- Queue and histogram
  - One million enqueue↔dequeue pairs / fill operations.
  - GCC TM
  - Intel TSX: Restricted Transactional Memory (RTM)
  - TinySTM
- Distribute work over 1-8 threads
- 10 warmup runs
- Take mean timing of 40 runs
- Regulate contention through a delay functor object
  - `LoadLevel::NONE` [0ns]
  - `LoadLevel::Low` [270ns]
  - `LoadLevel::Medium` [684ns]
  - `LoadLevel::High` [1554ns]

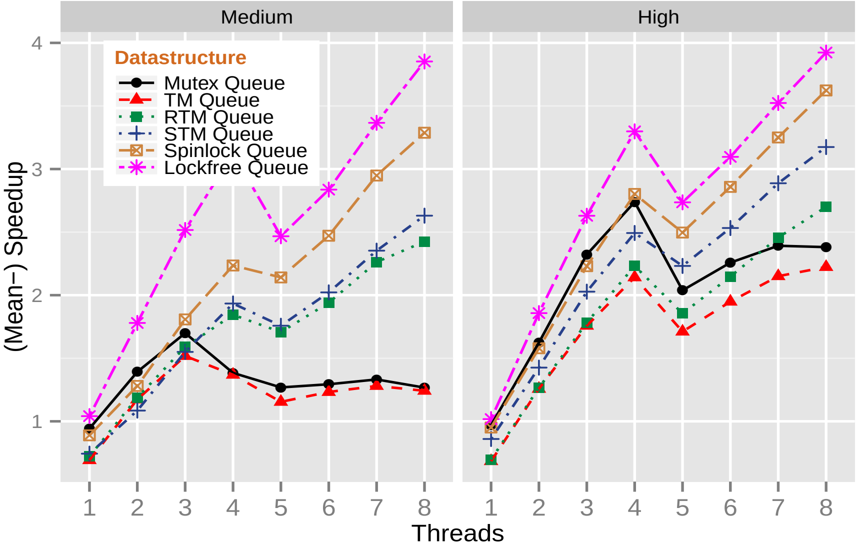
Transactional Memory

Queue Benchmark



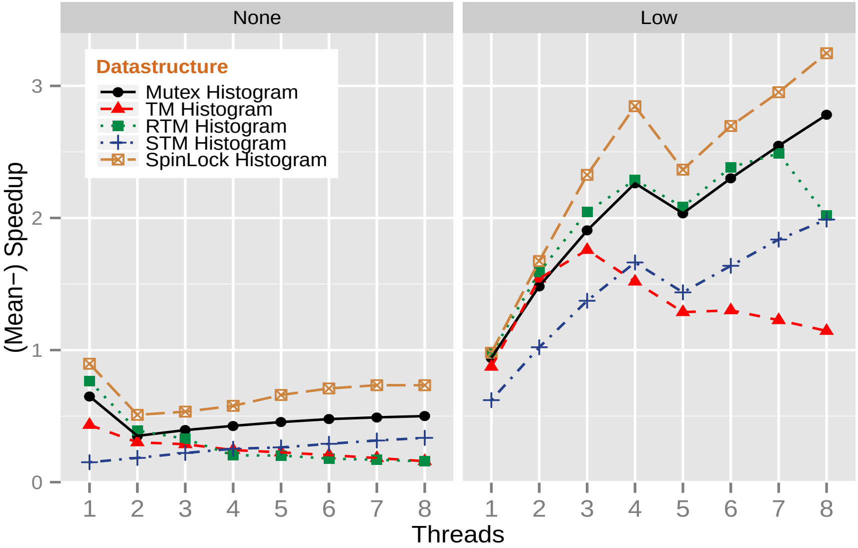
Transactional Memory

Queue Benchmark



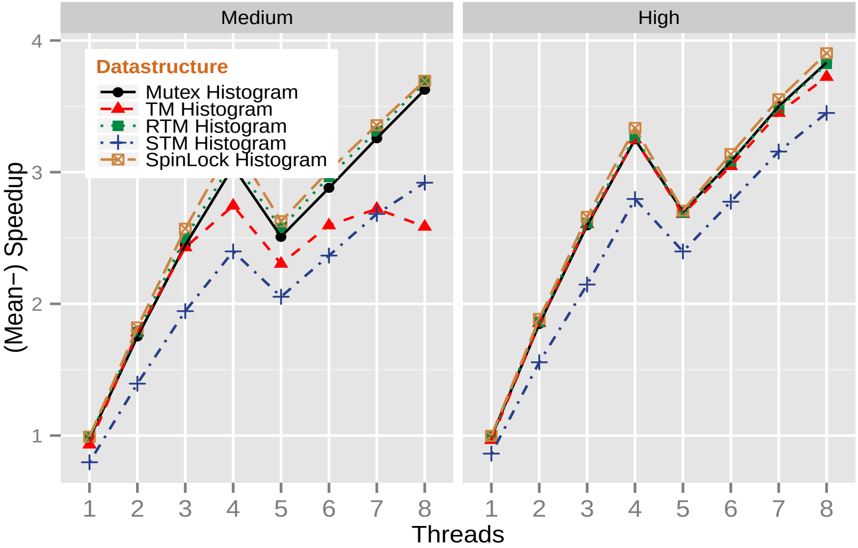
Transactional Memory

Histogram Benchmark



Transactional Memory

Histogram Benchmark



Transactional Memory

Experimental Evaluation in Literature

- Experimental evaluation of TM, especially hardware TM is rare
- No common conclusion has been drawn w.r.t. its feasibility
- Benchmark suite Lee-TM
  $\Rightarrow$ authors observe STM on par with coarse-grained locking ▸ Paper
- In general, STM is not outperforming conventional locking techniques
- "Peformance Evaluation of Intel TSX for High-Performance Computing" ▸ Paper
  - Sometimes outperforms even fine-grained locking solutions
  - But it sometimes performs worse than STM, when not optimized

Conclusion and Outlook

Conclusion and Outlook

- TM feasible?
  - As usual: it depends...
- Mutexes: Spend more time on debugging
- TM: Spend more time on making code faster
- New hardware implementations may improve performance
- Wait for C++ language extension and transaction safe STL

Thank you for your attention