Transactional Memory

*How to live without locks.*

**Till Smejkal**

3rd July 2018
Motivation

```c
int data[SIZE];

void thread_fn(int cnt) {
    /* initialize random */

    for (int i = 0; i < cnt; ++i) {
        auto pos = rand();
        data[pos]++;
    }
}
```
Motivation
Motivation

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Thread 1
Motivation

Thread 1

0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
Motivation

Thread 1
Motivation
Motivation

Thread 1

Thread 2
Motivation
Influential OS Research

Transactional Memory

Motivation

Thread 1

Thread 2
Motivation

Thread 1

Thread 2
Motivation

Coarse Locking

```c
int data[SIZE];
std::mutex mtx;

void thread_fn(int cnt) {
    /* initialize random */

    for (int i = 0; i < cnt; ++i) {
        auto pos = rand();

        mtx.lock();
        data[pos]++;
        mtx.unlock();
    }
}
```
Motivation

Coarse Locking
Motivation

Coarse Locking
Motivation

Coarse Locking
Motivation

Coarse Locking

| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Thread 1

Thread 2
Motivation

Coarse Locking

Thread 1

Thread 2
Motivation

Coarse Locking
Motivation

Coarse Locking
Motivation

Coarse Locking
Motivation

Coarse Locking
Motivation

Coarse Locking
Motivation
Coarse Locking

Thread 1

Thread 2
Motivation

Coarse Locking

Unfortunately, coarse-grained locking does not scale!

**Figure 1**: Time to access a shared data structure using coarse-grained locks.
Motivation
Fine Locking

```c
int data [ SIZE ];
std :: mutex mtx [ SIZE ];

void thread_fn ( int cnt ) {
    /* initialize random */
    for ( int i = 0; i < cnt; ++ i ) {
        auto pos = rand ( );
        mtx [ pos ]. lock ( );
        data [ pos ]++;
        mtx [ pos ]. unlock ( );
    }
}
```
Motivation

Fine Locking

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Thread 1   Thread 2
Motivation

Fine Locking
Motivation

Fine Locking
Influential OS Research

Transactional Memory

Motivation

Fine Locking

Thread 1

Thread 2
Motivation

Fine Locking
Motivation

Fine Locking
Motivation

Fine Locking

Thread 1

Thread 2
Motivation

Fine Locking

Thread 1

Thread 2

0 1 0 1 0 0 0 2 0 0 0 0 0 0 0 0 0
Motivation

Fine Locking

Fine-grained locks scale much better.

**Figure 2:** Time to access a shared data structure using coarse-grained or fine-grained locks.
Motivation

Fine Locking – Deadlock

Fine-grained locks become difficult if multiple locked elements have to be accessed.

```c
int data[SIZE];
std::mutex mtx[SIZE];

void thread_fn(int cnt) {
    /* initialize random */
    for (int i = 0; i < cnt; ++i) {
        auto pos1 = rand(), pos2 = rand();

        mtx[pos1].lock();
        mtx[pos2].lock();
        data[pos1] = 2*(data[pos2] + 1);
        mtx[pos2].unlock();
        mtx[pos1].unlock();
    }
}
```
Motivation
Fine Locking – Deadlock

Thread 1

Thread 2
Motivation

Fine Locking – Deadlock

Thread 1

Thread 2
Motivation

Fine Locking – Deadlock

Thread 1

Thread 2
Motivation

Fine Locking – Deadlock
Motivation

Fine Locking – Deadlock

Thread 1

Thread 2
Motivation

Fine Locking – Deadlock

```
0 2 0 0 0 0 0 0 2 0 0 0 0 0 0
```

Thread 1

Thread 2
Motivation

Fine Locking – Deadlock

Thread 1

Thread 2
Motivation

Fine Locking – Deadlock

Thread 1

Thread 2
Motivation

Fine Locking – Deadlock

Thread 1

Thread 2
Motivation

Fine Locking – Deadlock

0 2 0 0 0 0 2 0 0 2 0 0 0 0 0

Thread 1

Thread 2
Motivation

Fine Locking – Deadlock

Thread 1

Thread 2
Motivation

Fine Locking – Deadlock

Thread 1

Thread 2
Motivation

Fine Locking – Deadlock

Thread 1

Thread 2
Motivation
Fine Locking – Deadlock

Thread 1

Thread 2
Motivation

Fine Locking – Deadlock

Thread 1

Thread 2
Motivation

Fine Locking – Deadlock

Thread 1

Thread 2
Motivation

Fine Locking – Deadlock

Thread 1

Thread 2
Motivation

Fine Locking – Deadlock

Thread 1

Thread 2
Motivation

Coarse Locks
+ easy to use
+ no deadlock problem
- do not scale well

Fine Locks
+ scale well
- deadlock problem
Motivation

**Coarse Locks**
+ easy to use
+ no deadlock problem
- do not scale well

**Fine Locks**
+ scale well
- deadlock problem

*Transactional Memory* tries to combine the advantages of coarse- and fine-grained locks without having their disadvantages.
Motivation

Transactional memory can provide a similar performance as fine-grained locking.

Figure 3: Time to access a shared data structure using coarse-grained locks, fine-grained locks or transactional memory.
Outline

Transactional Memory

Hardware Transactional Memory
- Herlihy and Moss
- IBM Blue Gene/Q and System z
- Intel TSX

Software Transactional Memory
Transactional memory provides strong *automicity* and *isolation* guarantees for a finite number of instructions.
Transactional Memory

Transactional memory provides strong *automicity* and *isolation* guarantees for a finite number of instructions.

- Transactions are started and committed by the programmer.

```c
begin_transaction();
/*
 * Critical Section
 */
commit_transaction();
```
Transactional Memory

Transactional memory provides strong *automicity* and *isolation* guarantees for a finite number of instructions.

- Transactions are started and committed by the programmer.
- Reads from shared variables are tracked in the *read set*.

```c
begin_transaction();
/* transactional read */
int bar = foo;

commit_transaction();
```
Transactional Memory

Transactional memory provides strong *automicity* and *isolation* guarantees for a finite number of instructions.

```
1    begin_transaction();
2    int bar = foo;
3    /* transactional write */
4    baz = bar * 2 − foo;
5
6
7
8    commit_transaction();
```

- Transactions are started and committed by the programmer.
- Reads from shared variables are tracked in the *read set*.
- Writes to shared variables are tracked in the *write set*. 
Transactional Memory

Transactional memory provides strong *automicity* and *isolation* guarantees for a finite number of instructions.

```plaintext
1 begin_transaction();
2 int bar = foo;
3 baz = bar * 2 - foo;
4 /* explicit abort */
5 if (bar == 0)
6   abort_transaction();
7
8 commit_transaction();
```

- Transactions are started and committed by the programmer.
- Reads from shared variables are tracked in the *read set*.
- Writes to shared variables are tracked in the *write set*.
- Transactions can be aborted explicitly.
Transactional Memory

Transactional memory provides strong *automicity* and *isolation* guarantees for a finite number of instructions.

• Transactions are started and committed by the programmer.
• Reads from shared variables are tracked in the *read set*.
• Writes to shared variables are tracked in the *write set*.
• Transactions can be aborted explicitly.
• Write-write and read-write conflicts are detected by the system.

```c
begin_transaction();
int bar = foo;
baz = bar * 2 - foo;
if (bar == 0)
    abort_transaction();
commit_transaction();
```
Transactional Memory implementations can have many different properties.
Transactional Memory

Transactional memory implementations can have many different properties.

- **eager vs. lazy conflict detection**
  - **eager**: Detect conflicts during the execution of the transaction.
  - **lazy**: Check during the commit operation if a conflict happened.
Transactional Memory

Transactional memory implementations can have many different properties.

- eager vs. lazy conflict detection
- undo log vs. write buffering vs. versioning
  - **undo log**: Directly write-through to memory and keep log.
  - **buffering**: Keep all writes in local buffer and write to memory during commit.
  - **versioning**: Maintain multiple versions of the same variable for each transaction.
Transactional Memory

Transactional memory implementations can have many different properties.

- eager vs. lazy conflict detection
- undo log vs. write buffering vs. versioning
- implicit vs. explicit transaction begin
  - implicit: Automatically start a transaction with the first transactional operation (read, write).
  - explicit: Only start a transaction at the occurrence of a special operation.
Transactional Memory

Transactional memory implementations can have many different properties.

- eager vs. lazy conflict detection
- undo log vs. write buffering vs. versioning
- implicit vs. explicit transaction begin
- conflict detection granularity (variables, pages, cache lines, ...)

Till Smejkal
Transactional Memory

Transactional memory implementations can have many different properties.

- eager vs. lazy conflict detection
- undo log vs. write buffering vs. versioning
- implicit vs. explicit transaction begin
- conflict detection granularity (variables, pages, cache lines, ...)
- best-effort vs. progress guaranty
  - **best-effort** No guaranty is given which transaction is aborted at a conflict.
  - **progress guaranty** Some guaranties are provided by the system about which transaction is aborted at a conflict.
Transactional Memory

Transactional memory implementations can have many different properties.

- eager vs. lazy conflict detection
- undo log vs. write buffering vs. versioning
- implicit vs. explicit transaction begin
- conflict detection granularity (variables, pages, cache lines, …)
- best-effort vs. progress guaranty
- real nesting vs. flattened nesting vs. no nesting
  - real: Nested transactions are treated as full-fledged transactions and can commit and abort independently.
  - flattened: Nested transactions are only virtual transactions. They commit and abort with the surrounding transaction.
Outline

Transactional Memory

Hardware Transactional Memory
- Herlihy and Moss
- IBM Blue Gene/Q and System z
- Intel TSX

Software Transactional Memory
Herlihy and Moss
M. Herlihy and E. Moss first proposed hardware transactional memory in 1993 [6].

Their idea was to integrate TM support into the processor.
M. Herlihy and E. Moss first proposed hardware transactional memory in 1993 [6].

Their idea was to integrate TM support into the processor

- Special transactional read- and write-instructions
  - LT Transactionally read the value of a shared variable.
  - LTX Same as LT but with an additional hint a write will follow soon.
  - ST Tentatively write a value to a shared variable.
M. Herlihy and E. Moss first proposed hardware transactional memory in 1993 [6].

Their idea was to integrate TM support into the processor:

- Special transactional read- and write-instructions
- Special instructions to manage the transaction state
  
  COMMIT  Finishes the currently running transaction.
  VALIDATE Check if the system detected a conflict with a different transaction.
  ABORT  Abort the current transaction.
**Hardware Transactional Memory**

*Herlihy and Moss*

M. Herlihy and E. Moss first proposed hardware transactional memory in 1993 [6].

Their idea was to integrate TM support into the processor:

- Special transactional read- and write-instructions
- Special instructions to manage the transaction state
- Transactions are implicitly started by a transactional read- or write-instruction.
M. Herlihy and E. Moss first proposed hardware transactional memory in 1993 [6].

Their idea was to integrate TM support into the processor

- Special transactional read- and write-instructions
- Special instructions to manage the transaction state
- Transactions are implicitly started by a transactional read- or write-instruction.
- Conflict detection is done eagerly but aborts must be done explicitly.
M. Herlihy and E. Moss first proposed hardware transactional memory in 1993 [6].

Their idea was to integrate TM support into the processor:

- Special transactional read- and write-instructions
- Special instructions to manage the transaction state
- Transactions are implicitly started by a transactional read- or write-instruction.
- Conflict detection is done eagerly but aborts must be done explicitly.
- Transactions can contain non-transactional operations.
Hardware Transactional Memory

Herlihy and Moss – Implementation

The cache-coherency protocol is used to detect conflicts between transactions.
The cache-coherency protocol is used to detect conflicts between transactions.

- Cache lines in the read set (LT) must not be in the Invalid state.
The cache-coherency protocol is used to detect conflicts between transactions.

- Cache lines in the *read set* (LT) **must not** be in the *Invalid* state.
- Cache lines in the *write set* (LTX or ST) **must not** be in the *Shared* or *Invalid* state.
The cache-coherency protocol is used to detect conflicts between transactions.

- Cache lines in the *read set* (LT) must not be in the *Invalid* state.
- Cache lines in the *write set* (LTX or ST) must not be in the *Shared* or *Invalid* state.

Speculative writes are buffered in a special *transactional write-buffer*. 
The cache-coherency protocol is used to detect conflicts between transactions.

- Cache lines in the *read set* (LT) **must not** be in the *Invalid* state.
- Cache lines in the *write set* (LTX or ST) **must not** be in the *Shared* or *Invalid* state.

Speculative writes are buffered in a special *transactional write-buffer*.

- Allows parallel write-out to memory on commit and fast discard on abort.
Hardware Transactional Memory

*Herlihy and Moss – Implementation*

The cache-coherency protocol is used to detect conflicts between transactions.

- Cache lines in the *read set* (LT) must not be in the *Invalid* state.
- Cache lines in the *write set* (LTX or ST) must not be in the *Shared* or *Invalid* state.

Speculative writes are buffered in a special *transactional write-buffer*.

- Allows parallel write-out to memory on commit and fast discard on abort.
- Transaction size is not limited by cache size and associativity.
Hardware Transactional Memory

*Herlihy and Moss – Results*

**Figure 4:** Performance results of the H&M-HTM implementation in the *Counting Benchmark* (a) and *Double-Linked List Benchmark* (b).
IBM Blue Gene/Q and System z
In the year 2012, IBM presented a hardware transactional memory implementation for their System z machines [9].

The implementation extends the existing CPUs to support HTM
In the year 2012, IBM presented a hardware transactional memory implementation for their System z machines [9].

The implementation extends the existing CPUs to support HTM:

- Special instructions to start, abort, and commit a transaction:
  - TBEGIN  Start a new best-effort transaction, potentially nested.
  - TEND    Commit the currently running transaction.
  - TABORT  Explicitly abort the current transaction.
In the year 2012, IBM presented a hardware transactional memory implementation for their System z machines [9].

The implementation extends the existing CPUs to support HTM

- Special instructions to start, abort, and commit a transaction
- All instructions within a transaction are executed speculatively.
In the year 2012, IBM presented a hardware transactional memory implementation for their *System z* machines [9].

The implementation extends the existing CPUs to support HTM

- Special instructions to start, abort, and commit a transaction
- All instructions within a transaction are executed speculatively.
- Conflict detection is based on the cache-coherency protocol.
In the year 2012, IBM presented a hardware transactional memory implementation for their System z machines [9].

The implementation extends the existing CPUs to support HTM

- Special instructions to start, abort, and commit a transaction
- All instructions within a transaction are executed speculatively.
- Conflict detection is based on the cache-coherency protocol.
- Speculative stores are buffered in a store cache for isolation.
Influential OS Research

Transactional Memory

Hardware Transactional Memory
IBM System z

In the year 2012, IBM presented a hardware transactional memory implementation for their System z machines [9].

The implementation extends the existing CPUs to support HTM

- Special instructions to start, abort, and commit a transaction
- All instructions within a transaction are executed speculatively.
- Conflict detection is based on the cache-coherency protocol.
- Speculative stores are buffered in a store cache for isolation.
- Conflicts and store cache overflows cause transaction aborts.
In the year 2012, IBM presented a hardware transactional memory implementation for their *System z* machines [9].

The implementation extends the existing CPUs to support HTM

- Special instructions to start, abort, and commit a transaction
- All instructions within a transaction are executed speculatively.
- Conflict detection is based on the cache-coherency protocol.
- Speculative stores are buffered in a *store cache* for isolation.
- Conflicts and store cache overflows cause transaction aborts.
- Non-recoverable instructions (e.g. IO), interrupts, and faults also trigger aborts.
Hardware Transactional Memory

IBM System z

IBM also integrated other advanced features in their HTM implementation.

- Support for transaction debugging
- Non-speculative stores (NTSTG) to write debug information.
- Suppress debugging interrupts while running in a transaction.
- A Transactional Diagnostic Block containing additional information at an abort.
- Random transaction aborts to test fallback path.
- Transaction progress guaranty
- A special instruction TBEGINC can be used to start small transactions whose commit is ensured by the CPU.
- At a conflict older transactions are preferred.
- Filtering of interrupts to prevent switches into the kernel at an interrupt occurrence while being in a transaction.
Hardware Transactional Memory

* IBM System z

IBM also integrated other advanced features in their HTM implementation.

- Support for transaction debugging
Hardware Transactional Memory

IBM System z

IBM also integrated other advanced features in their HTM implementation.

- Support for transaction debugging
  - Non-speculative stores (NTSTG) to write debug information.
Hardware Transactional Memory

*IBM System z*

IBM also integrated other advanced features in their HTM implementation.

- Support for transaction debugging
  - Non-speculative stores (NTSTG) to write debug information.
  - Suppress debugging interrupts while running in a transaction.
Hardware Transactional Memory

IBM System z

IBM also integrated other advanced features in their HTM implementation.

- Support for transaction debugging
  - Non-speculative stores (NTSTG) to write debug information.
  - Suppress debugging interrupts while running in a transaction.
  - A *Transactional Diagnostic Block* containing additional information at an abort.
Hardware Transactional Memory

*IBM System z*

IBM also integrated other advanced features in their HTM implementation.

- Support for transaction debugging
  - Non-speculative stores (NTSTG) to write debug information.
  - Suppress debugging interrupts while running in a transaction.
  - A *Transactional Diagnostic Block* containing additional information at an abort.
  - Random transaction aborts to test fallback path.
Hardware Transactional Memory

*IBM System z*

IBM also integrated other advanced features in their HTM implementation.

- Support for transaction debugging
  - Non-speculative stores (NTSTG) to write debug information.
  - Suppress debugging interrupts while running in a transaction.
  - A *Transactional Diagnostic Block* containing additional information at an abort.
  - Random transaction aborts to test fallback path.
- Transaction progress guaranty
Hardware Transactional Memory

IBM System z

IBM also integrated other advanced features in their HTM implementation.

• Support for transaction debugging
  • Non-speculative stores (NTSTG) to write debug information.
  • Suppress debugging interrupts while running in a transaction.
  • A Transactional Diagnostic Block containing additional information at an abort.
  • Random transaction aborts to test fallback path.

• Transaction progress guaranty
  • A special instruction TBEGINC can be used to start small transactions whose commit is ensured by the CPU.
Hardware Transactional Memory
IBM System z

IBM also integrated other advanced features in their HTM implementation.

- Support for transaction debugging
  - Non-speculative stores (NTSTG) to write debug information.
  - Suppress debugging interrupts while running in a transaction.
  - A *Transactional Diagnostic Block* containing additional information at an abort.
  - Random transaction aborts to test fallback path.

- Transaction progress guaranty
  - A special instruction TBEGINC can be used to start small transactions whose commit is ensured by the CPU.
  - At a conflict older transactions are preferred.
Hardware Transactional Memory

IBM System z

IBM also integrated other advanced features in their HTM implementation.

- Support for transaction debugging
  - Non-speculative stores (NTSTG) to write debug information.
  - Suppress debugging interrupts while running in a transaction.
  - A Transactional Diagnostic Block containing additional information at an abort.
  - Random transaction aborts to test fallback path.
- Transaction progress guaranty
  - A special instruction TBEGINC can be used to start small transactions whose commit is ensured by the CPU.
  - At a conflict older transactions are preferred.
- Filtering of interrupts to prevent switches into the kernel at an interrupt occurrence while being in a transaction.
Hardware Transactional Memory

IBM System z – Results

**Figure 5**: Performance results of the IBM System z HTM when accessing four variables from a pool with 1k/10k elements.
Figure 6: Performance results of the IBM System z HTM when accessing four variables from a pool with 10 elements.
Hardware Transactional Memory

*IBM System z – Results*

**Figure 7:** Performance results of the IBM System z HTM when accessing one variable from a pool with 10 elements.
Hardware Transactional Memory
*IBM Blue Gene/Q*

The IBM *Blue Gene/Q* system was also extended by a HTM implementation in 2012 [12].

The implementation is similar to the System z one but
The IBM Blue Gene/Q system was also extended by a HTM implementation in 2012 [12].

The implementation is similar to the System z one but
- Speculative stores are buffered in the *multi-version* L2-Cache.
The IBM *Blue Gene/Q* system was also extended by a HTM implementation in 2012 [12].

The implementation is similar to the System z one but

- Speculative stores are buffered in the *multi-version L2-Cache*.
- Transactions either bypass the shared L1-Cache or are isolated from the other cores using *cache coloring*. 
Hardware Transactional Memory

*IBM Blue Gene/Q*

The IBM *Blue Gene/Q* system was also extended by a HTM implementation in 2012 [12].

The implementation is similar to the System z one but

- Speculative stores are buffered in the *multi-version L2-Cache*.
- Transactions either bypass the shared L1-Cache or are isolated from the other cores using *cache coloring*.
- Conflicts are detected by an additional logic in the L2-Cache.
The IBM Blue Gene/Q system was also extended by a HTM implementation in 2012 [12].

The implementation is similar to the System z one but

- Speculative stores are buffered in the multi-version L2-Cache.
- Transactions either bypass the shared L1-Cache or are isolated from the other cores using cache coloring.
- Conflicts are detected by an additional logic in the L2-Cache.
- Only best-effort transactions are implemented in hardware.
Hardware Transactional Memory

IBM Blue Gene/Q

The Blue Gene/Q HTM additionally provides compiler support and a specialized runtime.
The Blue Gene/Q HTM additionally provides compiler support and a specialized runtime.

- Software support for transaction retrying and forward progress guaranty.
- Automatic fallback to using locks

```c
void thread_fn(int cnt) {
    for (int i = 0; i < cnt; ++i) {
        auto pos = rand();
        #pragma tm_atomic {
            data[pos]++;
        }
    }
}
```
There are four main benefits of the IBM Blue Gene/Q HTM: 

- **Software support for transaction retrying and forward progress guaranty.**
- **Automatic fallback to using locks.**
- **Compiler extensions allowing easy integration of transactional code segments.**

```c
void thread_fn(int cnt) {
    /* initialize random */
    for (int i = 0; i < cnt; ++i) {
        auto pos = rand();
        #pragma tm_atomic
            data[pos]++;
    }
}
```
Figure 8: Performance results of the IBM Blue Gene/Q HTM for various benchmarks of the STAMP [10] suite.
Intel Transactional Synchronization Extension
Hardware Transactional Memory

*Intel TSX*

Since the Haswell processor generation Intel provides a HTM implementation for consumer systems [8, 13].

Intel’s HTM comes with two distinct features
Since the Haswell processor generation Intel provides a HTM implementation for consumer systems [8, 13].

Intel’s HTM comes with two distinct features

• Hardware Lock Elision (HLE)
Hardware Transactional Memory

*Intel TSX*

Since the Haswell processor generation Intel provides a HTM implementation for consumer systems [8, 13].

Intel’s HTM comes with two distinct features

- Hardware Lock Elision (HLE)
  - Backward compatible hardware extension to automatically replace locks with transactions.
Hardware Transactional Memory

*Intel TSX*

Since the Haswell processor generation Intel provides a HTM implementation for consumer systems [8, 13].

Intel’s HTM comes with two distinct features

- Hardware Lock Elision (HLE)
  - Backward compatible hardware extension to automatically replace locks with transactions.
- Restricted Transactional Memory (RTM)
Hardware Transactional Memory

Intel TSX

Since the Haswell processor generation Intel provides a HTM implementation for consumer systems [8, 13].

Intel’s HTM comes with two distinct features

- **Hardware Lock Elision (HLE)**
  - Backward compatible hardware extension to automatically replace locks with transactions.

- **Restricted Transactional Memory (RTM)**
  - Full featured best-effort transactional memory implementation.
Since the Haswell processor generation Intel provides a HTM implementation for consumer systems [8, 13].

Intel’s HTM comes with two distinct features

• Hardware Lock Elision (HLE)
  • Backward compatible hardware extension to automatically replace locks with transactions.

• Restricted Transactional Memory (RTM)
  • Full featured best-effort transactional memory implementation.
  • Not backward compatible to older processors.
Hardware Transactional Memory

Intel TSX – HLE

HLE can be used to easily transform locked regions into transactions.
Hardware Transactional Memory

Intel TSX – HLE

HLE can be used to easily transform locked regions into transactions.

- Prefix lock operations with XACQUIRE and XRELEASE
HLE can be used to easily transform locked regions into transactions.

- Prefix lock operations with XACQUIRE and XRELEASE
- Hardware will try to execute the region as transaction without taking the lock.
Hardware Transactional Memory
*Intel TSX – HLE*

HLE can be used to easily transform locked regions into transactions.

- Prefix lock operations with XACQUIRE and XRELEASE
- Hardware will try to execute the region as transaction without taking the lock.
- If the transaction aborts, the CPU will automatically retry and take the lock.
HLE can be used to easily transform locked regions into transactions.

- Prefix lock operations with XACQUIRE and XRELEASE
- Hardware will try to execute the region as transaction without taking the lock.
- If the transaction aborts, the CPU will automatically retry and take the lock.
- Older processors ignore the prefix and directly acquire the lock.
Hardware Transactional Memory

*Intel TSX – RTM*

Intel RTM is a best-effort hardware transactional memory implementation with flattened nesting support.
Hardware Transactional Memory

Intel TSX – RTM

Intel RTM is a best-effort hardware transactional memory implementation with flattened nesting support.

- Conflict detection is based on the cache-coherency protocol.
Hardware Transactional Memory
*Intel TSX – RTM*

Intel RTM is a best-effort hardware transactional memory implementation with flattened nesting support.

- Conflict detection is based on the cache-coherency protocol.
- Speculative operations are isolated in the L1-Cache of the core.
Hardware Transactional Memory

*Intel TSX – RTM*

Intel RTM is a best-effort hardware transactional memory implementation with flattened nesting support.

- Conflict detection is based on the cache-coherency protocol.
- Speculative operations are isolated in the L1-Cache of the core.
- Speculative reads can be evicted to the L2-Cache without transactional abort.
Intel RTM is a best-effort hardware transactional memory implementation with flattened nesting support.

- Conflict detection is based on the cache-coherency protocol.
- Speculative operations are isolated in the L1-Cache of the core.
- Speculative reads can be evicted to the L2-Cache without transactional abort.
- Transactions must be started and stopped explicitly by the programmer (XBEGIN and XEND).

Till Smejkal
Hardware Transactional Memory

*Intel TSX – RTM*

Intel RTM is a best-effort hardware transactional memory implementation with flattened nesting support.

- Conflict detection is based on the cache-coherency protocol.
- Speculative operations are isolated in the L1-Cache of the core.
- Speculative reads can be evicted to the L2-Cache without transactional abort.
- Transactions must be started and stopped explicitly by the programmer (**XBEGIN** and **XEND**).
- Conflicts cause implicit aborts of transactions.
Hardware Transactional Memory

*Intel TSX – RTM*

Intel RTM is a best-effort hardware transactional memory implementation with flattened nesting support.

- Conflict detection is based on the cache-coherency protocol.
- Speculative operations are isolated in the L1-Cache of the core.
- Speculative reads can be evicted to the L2-Cache without transactional abort.
- Transactions must be started and stopped explicitly by the programmer \((XBEGIN\) and \(XEND\)).
- Conflicts cause implicit aborts of transactions.
- Interrupts, irrecoverable operations, and faults also trigger aborts.
Hardware Transactional Memory

*Intel TSX – Example*

```cpp
int data[SIZE];

void thread_fn(int cnt) {
    /* initialize random */

    for (int i = 0; i < cnt; ++i) {
        auto pos = rand();

        while (true) {
            if (_xbegin() == _XBEGIN_STARTED) {
                data[pos]++;
                _xend();
                break;
            }
        }
    }
}
```
Hardware Transactional Memory

*Intel TSX – Results*

**Figure 9:** Performance results of the Intel HTM for various benchmarks of the STAMP [10] suite.
Outline

Transactional Memory

Hardware Transactional Memory
- Herlihy and Moss
- IBM Blue Gene/Q and System z
- Intel TSX

Software Transactional Memory
Software Transactional Memory

Instead of relying on hardware, transactional memory can also be implemented in software [11].
Software Transactional Memory

Instead of relying on hardware, transactional memory can also be implemented in software [11].

- Runtime intercepts transactional reads and writes and tracks dependencies and conflicts.
Software Transactional Memory

Instead of relying on hardware, transactional memory can also be implemented in software [11].

- Runtime intercepts transactional reads and writes and tracks dependencies and conflicts.
- Various implementations exist using *undo-logs* or *shadow-copies*.
Instead of relying on hardware, transactional memory can also be implemented in software [11].

- Runtime intercepts transactional reads and writes and tracks dependencies and conflicts.
- Various implementations exist using *undo-logs* or *shadow-copies*.
- Conflicts are detected lazily in most implementations.
Software Transactional Memory

Instead of relying on hardware, transactional memory can also be implemented in software [11].

- Runtime intercepts transactional reads and writes and tracks dependencies and conflicts.
- Various implementations exist using **undo-logs** or **shadow-copies**.
- Conflicts are detected lazily in most implementations.
- Requires intensive compiler and runtime support.
Instead of relying on hardware, transactional memory can also be implemented in software [11].

- Runtime intercepts transactional reads and writes and tracks dependencies and conflicts.
- Various implementations exist using *undo-logs* or *shadow-copies*.
- Conflicts are detected lazily in most implementations.
- Requires intensive compiler and runtime support.

Many different implementations of STM exist [5, 7, 1, 2] whereas their performance differs significantly [3, 4].
Any Questions?
References I


- Aleksandar Dragojevic et al. “Why STM can be more than a research toy”. In: Communications of the ACM 54.4 (2011), pp. 70–77.


References II


Assignments

Available Topics:

• Distributed Resource Management – Michael
• Transactional Memory – Till
• Robustness in File Systems – Carsten
• Trusted Execution Environments – Carsten/Michael
• Architecture Simulation – Matthias
• Resilience and Fault Tolerance – Matthias
• UNIX-like Systems – Nils
• Distributed Debugging – Maksym
• Performance Modeling – Maksym
• Attacks on SGX – Jan
• HPC vs. Cloud Computing – Jan