Intel TSX

Adam Dees

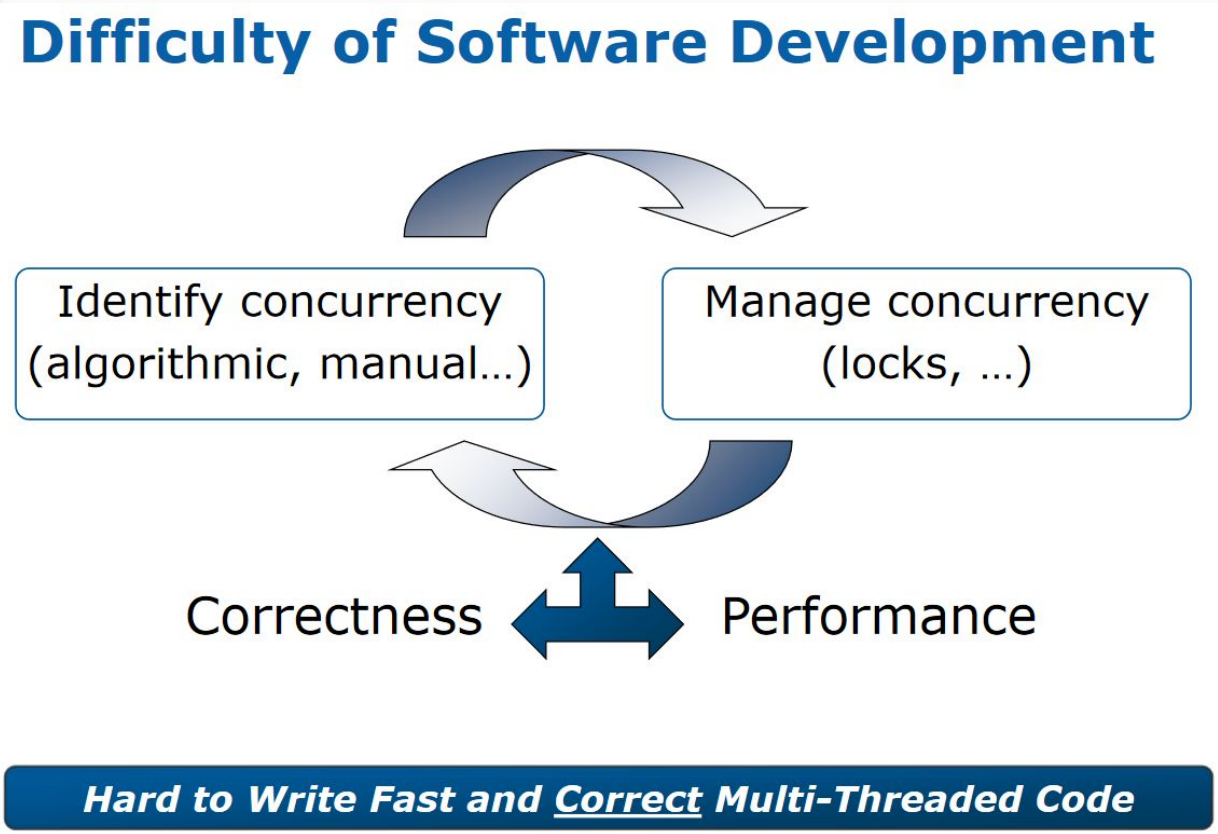
Intel TSX/TSX-NI

Transactional Synchronization Extensions / Transactional Synchronization Extensions New Instructions

- Provides hardware level transactional memory
- Expands Intel's x86 Instruction Set Architecture
- Released in 2013
- Two different implementations: Hardware Lock Elision (HLE) and Restricted Transactional Memory (RTM)

Motivation

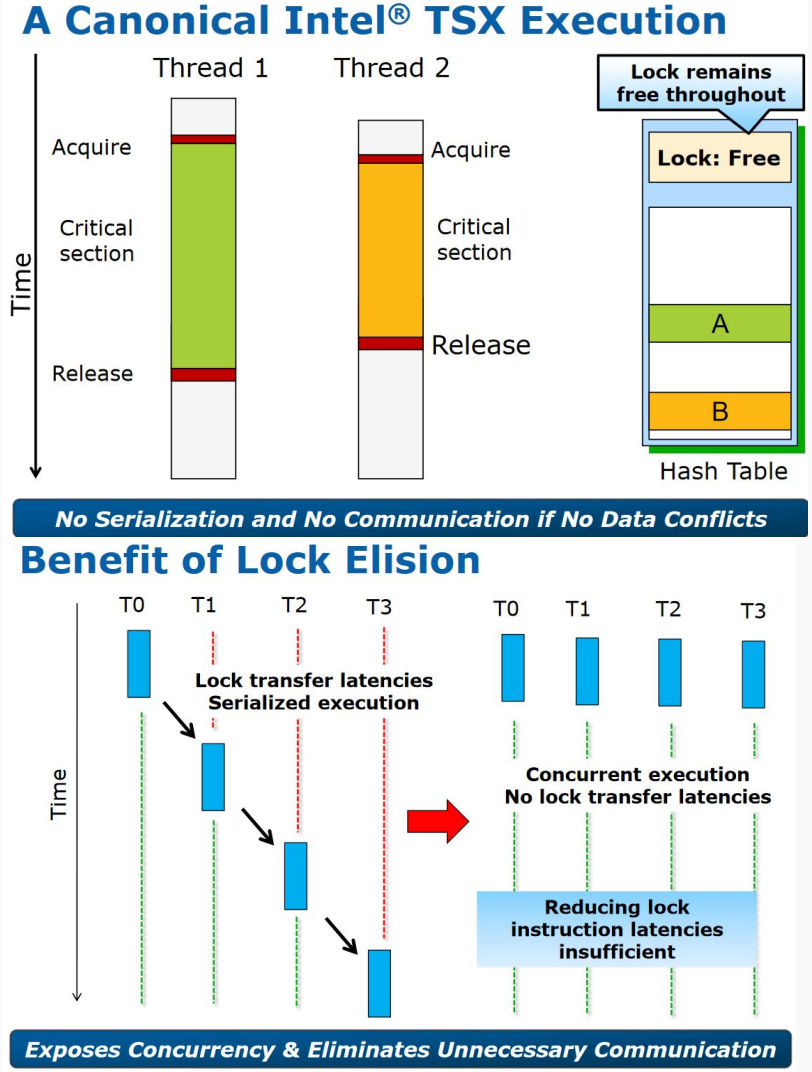
- Writing multithreaded code is difficult or at the very least, time consuming
- The promise of transactional memory is the ability to write parallel code easily
- We want to sacrifice as little efficiency as possible



From Intel's presentation on "Transactional Synchronization Extensions"

Lock Elision

- An elision is an omission, in this case the omission of writing to a lock
- You might call running multiple critical sections without any lock writing optimistic at best
- So it fits that this is a kind of ‘optimistic concurrency control’
- Saves lots of time that would otherwise be spent synchronizing
- We watch out for conflict and retrace our work if it occurs
- A very natural way to exploit unspecified concurrency



From Intel’s presentation on “Transactional Synchronization Extensions”

Hardware Lock Elision (HLE) vs. Restricted Transactional Memory (RTM)

HLE

- Has backwards compatibility with some prior processors
- XACQUIRE and XRELEASE
- These denote the start and end of a critical section
- Only a subset of instructions will work in these critical code sections
- Critical section failure leads to re-execution without lock elision

RTM

- Processor must provide explicit support for RTM
- XBEGIN, XEND, and XABORT
- The XBEGIN instruction includes a redirect to another section of code
- If the transaction fails, we move to this section of code and update a special register called EAX with an encoding that specifies the cause of failure

XTEST is found in both, it tests if you are inside of a transaction

Intel® TSX Interface: HLE

```
        mov eax, 1
Try:    lock xchg mutex, eax
        cmp eax, 0
        jz Success
Spin:   pause
        cmp mutex, 1
        jz Spin
        jmp Try
```

```
        mov eax, 1
Try:    xacquire lock xchg mutex, eax
        cmp eax, 0
        jz Success
Spin:   pause
        cmp mutex, 1
        jz Spin
        jmp Try
```

Enter HLE execution

If lock not free, execution will abort either early (if pause used) or when lock gets free

Library

Application

```
acquire_lock (mutex)
; do critical section
; function calls,
; memory operations, …
release_lock (mutex)
```

```
mov mutex, 0
```

```
xrelease mov mutex, 0
```

Commit HLE execution

Intel® TSX Interface: RTM

```
Retry: xbegin Abort
       cmp mutex, 0
       jz Success
       xabort $0xff

Abort:
    … check EAX and do retry policy
    … actually acquire lock or wait
    … to retry.
    …
```

… Enter RTM execution, Abort is fallback path

… Check to see if mutex is free

… Abort transactional execution if mutex busy

… Fallback path in software

… Retry RTM or explicitly acquire mutex

```
acquire_lock (mutex)
; do critical section
; function calls,
; memory operations, …
release_lock (mutex)
```

```
cmp mutex, 0
jnz release_lock
xend
```

… Mutex not free →was not an RTM execution

… Commit RTM execution

Conflict Detection

- A big open question at this point is how we can detect conflicts
- The specification is not fully given, but we know Intel maintains a read and write-set for each transaction. The sets are sets of caches. So cache-level is the granularity of this detection
- From Intel's manual: "A conflicting data access occurs if another logical processor either reads a location that is part of the transactional region's write-set or writes a location that is a part of either the read- or write-set of the transactional region. We refer to this as a data conflict."
- "Transactional aborts may also occur due to limited transactional resources"
- At the end of the day with such generality, we may be aborting constantly and it will be hard to know until the program runs

Details on Conflict Detection and Contention Management

- In hardware, read and write sets will be appended to cache lines as a RS bit and WS bit
- Cache controller is used to detect these conflicts
- When a conflict is detected by a transaction, the transaction itself aborts
    - Therefore old gives way to new
- This is done because of the desire to preserve usage of the cache coherence protocol, which does the same

Each transaction keeps track of a read and write set like so:

Cache lines	RS	WS
. . .	0	0
. . .	1	0
. . .	1	1

Aborts and Commits

- A copy of the register is made at the start of the transaction
- To abort, the register is restored to its original state, WS/RS bits are zeroed out, and all WS lines are put in an invalid state
- A commit places all WS lines in M state, RS in S/F state, all WS/RS bits are zeroed, and the original copy of the register is removed such that the existing register contents are the new architectural state

Does Intel TSX accomplish its goals?

- “Performance Evaluation of Intel Transactional Synchronization Extensions for High-Performance Computing” says that TSX . . .
  - “on a set of real-world, high-performance computing workloads, Intel TSX provides 1.41x average speedup over lock and atomics based implementations”
  - “we observe an average of 1.31x bandwidth improvement on a set of network intensive applications”
- A significant improvement on performance
- Also makes the code easier/faster to write, so win/win
- However . . .

Security Issues, Removal, and Revival?

- “Breaking Kernel Address Space Layout Randomization with Intel TSX” in 2016 showed exactly what its title suggests
- Address space layout randomization is a technique to prevent vulnerabilities in memory by making attacks impossible to reliably reach a particular section of memory
- Because of how Intel TSX handles aborts, it is possible for attackers to reliably discover the location of an otherwise randomly placed kernel address space on all operating systems
- HLE has been taken out of all Intel processors from 2019 and later
- Intel 10th generation processors do not support TSX at all, neither HLE or RTM
- A new TSX-like TSXLDTRK instruction set extension has been documented and planned for inclusion in some future server processors

Slides borrowed from the “Intel®Transactional Synchronization Extensions” presentation given at the Intel Developer Forum 2012. http://pages.cs.wisc.edu/~rajwar/papers/sf12_arcs004_100.pdf