VMM Emulation of Intel Hardware Transactional Memory

Maciej Swiech, Kyle Hale, Peter Dinda
Northwestern University
V3VEE Project
www.v3vee.org

Hobbes Project
What will we talk about?

• We added the capability to run Intel HTM code on a virtual machine with minimal emulation

• We developed a new page-flipping technique that allows capturing of reads and writes at single memory reference granularity

• Software implementation of HTM emulation allows for arbitrary transaction size and code testing
Outline

• Motivation / Background
• Intel HTM
• Architecture
• Palacios
• Evaluation
• Conclusions
Outline

• Motivation / Background
  • Intel HTM
  • Architecture
  • Palacios
• Evaluation
• Conclusions
Motivation | transactional memory

• Processors and applications become more parallel and distributed to cope with growing scale of data and research problems

• Need for easier and more reliable methods for concurrent programming
do_the_things();

```cpp
void do_the_things() {
    write_shared_mem();
    read_shared_mem();
}
```
Background | transactional memory

Instead of:

```c
acquire_lock();

do_the_things();

release_lock();
```

```c
7
do_the_things() {
    write_shared_mem();
    read_shared_mem();
}
```
Background | transactional memory

Instead of:

```c
acquire_lock();
do_the_things();
release_lock();
```

Have to track locks

Deadlock
Background | transactional memory

acquire_lock();
do_the_things();
release_lock();

Can do:

transaction {
    do_the_things();
}
Background | transactional memory

Unsafe concurrent memory accesses are detected by TM
Easier to write safe code

Can do:

```c
acquire_lock();

do_the_things();

release_lock();
```

UNSAFE:
Write after Read
Read after Write
Write after Write
Background | transactional memory

• Transactions are
  • Composable
  • Easier to reason about
  • More optimistic than locking
    • Assumption: no other code will touch memory in TX

• HTM is faster than STM
Motivation | virtualizing

- Currently only Intel Haswell and IBM chipsets have implementations of Hardware Transactional Memory

- Adding HTM capabilities to a virtual machine monitor would allow anyone to run transactional code

- Allows for testing effects of new hardware implementations on code
Outline

• Motivation / Background
• Intel HTM
• Architecture
• Palacios
• Evaluation
• Conclusions
Intel HTM | background

• In the Haswell generation of processors Intel introduced 2 Hardware Transactional Memory implementations
  • RTM – Restricted Transactional Memory
  • HLE – Hardware Lock Elision

• 4 new instructions added to the ISA
  • XBEGIN
  • XABORT
  • XEND
  • XTEST
Intel HTM | ISA

- **XBEGIN imm32**
  - Marks beginning of a transaction and abort label

- **XABORT imm32**
  - Forces transaction abort

- **XEND**
  - Marks end of transaction

- **XTEST**
  - Tests if processor is currently in a transaction state
start_label:
   XBEGIN abort_label
   <body of transaction, may use XABORT>
   XEND
success_label:
   <handle transaction committed>
abort_label:
   <handle transaction aborted>
Intel HTM | specification

• Intel list many reasons a transaction “may” abort
  • Operations that modify RIP, GPRs, status flags
  • Operations on XMM, YMM, MXCSR registers
  • Various other instructions
  • Synchronous exception events
  • Asynchronous events such as interrupts
  • Self-modifying code
  • Many others...

• RaW, WaR, WaW conflicts trigger an abort
Outline

• Motivation / Background
• Intel HTM
• Palacios
• Architecture
• Evaluation
• Conclusions
Architecture | design

- Hypervisor extension
  - TM events captured and handled in VMM
- Redo-log based design with garbage collection
- Minimal instruction decoding
Architecture | design

- MIME
  - Generate stream of memory read/writes

- RTME
  - Maintains the redo log
  - Tracks system state

- Conflict Detection

- Garbage Collection
Architecture | RTME

• Finite State Machine model
  • SYSTEM state
  • CORE state

• TSX instructions generate #UD exceptions, driving state

• Maintains read/write logs for each transaction
Architecture | RTME

• Keeps track of per-core and system transactional state

• Places cores in single-stepping mode
  • If one core single-stepping, all cores

• Launches garbage collection of log entries
Architecture | example

Restricted Transactional Memory Engine

start_label:
  XBEGIN abort_label
  <body of transaction, may use XABORT>
  XEND
success_label:
  <handle transaction committed>
abort_label:
  <handle transaction aborted>
start_label:

XBEGIN abort_label

<body of transaction, may use XABORT>

XEND

success_label:

<handle transaction committed>

abort_label:

<handle transaction aborted>
Architecture | example

Restricted Transactional Memory Engine

start_label:
    XBEGIN abort_label
    <body of transaction, may use XABORT>
    XEND
success_label:
    <handle transaction committed>
abort_label:
    <handle transaction aborted>

Monitor abort conditions (incl. XABORT)
Maintain redo-log
Architecture | example  Restricted Transactional Memory Engine

```
start_label:
    XBEGIN abort_label
    <body of transaction, may use XABORT>
    XEND

success_label:
    <handle transaction committed>

abort_label:
    <handle transaction aborted>
```

CHECK WaW conflicts
CHECK RaW conflicts
CHECK WaR conflicts
start_label:
    XBEGIN abort_label
    <body of transaction, may use XABORT>
    XEND
success_label:
<handle>
abort_label:
<handle transaction aborted>
start_label:
  XBEGIN abort_label
  <body of transaction, may use XABORT>
  XEND
success_label:
  <handle transaction committed>
abort_label:
  <handle transaction aborted>
Architecture | example

Restricted Transactional Memory Engine

start_label:

XBEGIN abort_label

<body of transaction, may use XABORT>

XEND

success_label:

<handle transaction committed>

abort_label:

<handle transaction aborted>

Core out of TM mode
Launch GC

if no cores in TM,
System out of TM mode
start_label:
    XBEGIN abort_label
    <body of transaction, may use XABORT>
    XEND
success_label:
    <handle transaction committed>
abort_label:
    <handle transaction aborted>

If any abort condition is triggered
Runs at given code point

All intermediate state is discarded
Architecture | MIME

• Leverages
  • Shadow Page Table page fault hooking
  • Instruction length decoding
  • Hypercall insertion

→ Memory access single-stepping

• Staging page to keep writes hidden until commit
prev: addq %rbx, %rax
cur:  INSTRUCTION
next: movq %rdx, %rbx
...
target:
...
prev: addq %rbx, %rax

cur:  INSTRUCTION

next: movq %rdx, %rbx

...  
target: 

...
prev:  addq %rbx, %rax
cur:   INSTRUCTION
next:  VMCALL
...   ...
target: ...
...   ...

saved instr:  movq %rdx, %rbx
Architecture | example

prev: addq %rbx, %rax

cur: INSTRUCTION

next: VMCALL

... target:

... saved instr: movq %rdx, %rbx

Flush the shadow page tables

All guest mem access → page fault
prev: addq %rbx, %rax
cur: INSTRUCTION
next: VMCALL
... target:
...
saved instr: movq %rdx, %rbx

IFETCH $\rightarrow$ sPT fault
Map the instruction page in
Architecture | example

prev: addq %rbx, %rax

cur:   Instruction

next: VMCALL

... 

target: 

... 

saved instr: movq %rdx, %rbx

Memory and Instruction Meta Engine

Read: map page in as read-only
Write: map staging page in

Read: record address
Write: record address and value
Architecture | example

prev:  addq  %rbx,  %rax

cur:   INSTRUCTION

next:  VMCALL

...  

target:  

...

Memory and Instruction Meta Engine

Signals end of instruction
If staging page was used, copy data into redo log

saved instr:  movq  %rdx,  %rbx
prev: addq %rbx, %rax
cur: INSTRUCTION
next: movq %rdx, rbx
... target:
... saved instr: NULL
addq %rbx, %rax

prev: INSTRUCTION

cur: movq %rdx, rbx

... target:

... saved instr: NULL

MIME begins again
prev: addq %rbx, %rax
cur:  INSTRUCTION
next: movq %rdx, rbx
...
target: VMCALL
...
saved instr: ...
Architecture | conflict checking

• All transactions are given a number, which serves as a context and gives the transactions an ordering

• 2 additional 2D hash tables are maintained
  • Record during which system state (TX number) memory accesses were made
  • Record if accesses were reads or writes
Architecture | conflict checking

• On a transaction end, every access in the RTME log is checked against the conflict tables

• If conflict is detected, transaction is aborted
Architecture | Garbage Collection

- Log entries and collision hashes will keep growing
  - Garbage collection is needed

- Garbage collection is launched at transaction end

- Transaction number context is monotonically increasing on each core
  - Easy to determine accesses made during contexts no longer referenced
Outline

• Motivation / Background
• Intel HTM
• Architecture
• Palacios
• Evaluation
• Conclusions
Palacios | background

• OS-independent, open source, BSD-licensed, publicly available embeddable VMM

• Collaborative community resource development project involving Northwestern University, the University of New Mexico, University of Pittsburgh, Sandia National Labs, and Oak Ridge National Lab

• Currently leveraged for Hobbes Node Virtualization Layer
Palacios |

- HTM implementation could be added to any hypervisor with shadow page table fault hooking
  - No instruction emulation necessary

- ~1300 lines of code

- RTME/MIME available as patchset
Outline

• Motivation / Background
• Intel HTM
• Architecture
• Palacios
• Evaluation
• Conclusions
Evaluation |

• RTME/MIME vs Intel Haswell

• RTME/MIME and Intel SDE vs ‘native’
Evaluation | performance

- HP Proliant DL320e
- 1x quad-core Intel Xeon E3-1720v3
- 8GB RAM.
- Fedora 20 with a 3.13.5 kernel
Evaluation | performance

• Microbenchmark
  • One thread pinned to a single core
  • Enters a transaction, writes to a memory location, and then exits the transaction.
  • Benchmark measures the time spent running 10 such transactions,
  • Runtime averaged over 100 runs.
Evaluation | performance

<table>
<thead>
<tr>
<th>HTM implementation</th>
<th>Average runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTME/MIME</td>
<td>853.88 usec</td>
</tr>
<tr>
<td>Intel Haswell</td>
<td>2.57 usec</td>
</tr>
</tbody>
</table>
Evaluation | performance

<table>
<thead>
<tr>
<th>HTM implementation</th>
<th>Average runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTME/MIME</td>
<td>853.88 usec</td>
</tr>
<tr>
<td>Intel Haswell</td>
<td>2.57 usec</td>
</tr>
</tbody>
</table>

Only during TX
~3% overhead otherwise
Evaluation | correctness

- Dell PowerEdge R415
- 2x quadcore AMD Opteron 4122 installed
- 16 GB of memory.
- Fedora 15 with a 2.6.38 kernel
- 2 virtual cores
- BusyBox environment based on Linux kernel 2.6.38
- This machine does not have an HTM implementation.
Evaluation | correctness

• Suite of micro-benchmarks
  • Transaction calls XABORT not having written to memory
  • Transaction calls XABORT after having “written” to memory
  • Transaction writes memory with an immediate value
  • Transaction reads memory into a register
  • Transaction writes a register to memory
  • Transaction reads and writes the same memory location
  • Transaction thread writes to distinct addresses
  • Transaction and non-transactional thread write to overlapping addresses.

• Threads written using pthreads
Evaluation | correctness

• All test-cases run on RTME/MIME and Intel SDE 5.31.0

• All test-cases run (without TSX instructions) on the host

<table>
<thead>
<tr>
<th>Emulation Method</th>
<th>Slowdown vs. Native</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTME/MIME</td>
<td>~1,500x</td>
</tr>
<tr>
<td>Intel SDE 5.31.0</td>
<td>~90,000x</td>
</tr>
</tbody>
</table>
Outline

• Motivation / Background
• Intel HTM
• Palacios
• Architecture
• Evaluation
• Conclusions
Conclusion

- Developed RTME/MIME system
  - Software implemented HTM emulation system

- Developed MIME
  - Novel page-flipping ‘single stepping’ technique

- Performance
  - Run significantly faster than emulation
Conclusion | limitations

• Page boundaries
  • No support for instructions or memory accesses that cross page boundaries

• Read-after-Write accesses
  • sPT hooking doesn’t allow detection of RaW
  • Fine for correctness of implementation

• REP prefix
  • No support for instructions with multiple accesses
Conclusion | future work

• Extend MIME
  • Leverage instruction recording to capture detailed memory traces of application runs
  • Include support for breakpoints / stack traces to aid with concurrent debugging tools
Conclusion | future work

- Leverage software cache to test limitations on transaction size
Acknowledgements

• NU EECS 441 HTM Team
  • Marcel Flores
  • Zachary Bischof

• Quix86 x86 decoder team
  • Alexander Kudryavtsev
  • Michael Solovyov
• HTM emulation with **minimal instruction emulation**
• Page-flipping technique for capturing memory accesses at **memory access granularity**
• Software controlled HTM emulation implementation for testing

---

**Maciej Swiech** <mswiech@u.northwestern.edu>

http://eecs.northwestern.edu/~msw978

Prescience Lab: www.presciencelab.org

V3VEE Project: www.v3vee.org