VMM-based Emulation of Intel Hardware Transactional Memory

Maciej Swiech, Kyle C. Hale, Peter A. Dinda
Department of EECS
Northwestern University
{m-swiech,k-hale,pdinda}@northwestern.edu

Intel TSX ISA

Our system adds the ability to run RTM extensions on hardware that does not have HTM, analyze behavior of HTM under various cache sizes, and supports arbitrary size transactions.

Hardware Transactional Memory (HTM) is a long-standing concept that holds considerable promise for improving the correctness and performance of concurrent programs on hardware multiprocessors. Intel has made HTM a component of its next generation of x86/64 platform, Haswell. We focus on the Restricted Transactional Memory (RTM) extensions to the x86 ISA – XBEGIN, XEND, XTEST, and XABORT. These 4 instructions define a transaction block. Various conditions, such as conflicting memory accesses, control flow instructions, and hardware and software interrupts can cause a transaction to be aborted. An RTM transaction typically has the following form:

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

System Architecture

RTME

Restricted Transactional Memory Engine

- Keeps track of transactional state for system, and for each core
- Manages redo log for each core
- Driven by #UD exceptions, conflict hash entries, internal cache model, and external events
- Drives conflict checking to ensure transaction atomicity
- Initiates garbage collection of conflict hash entries

MIME

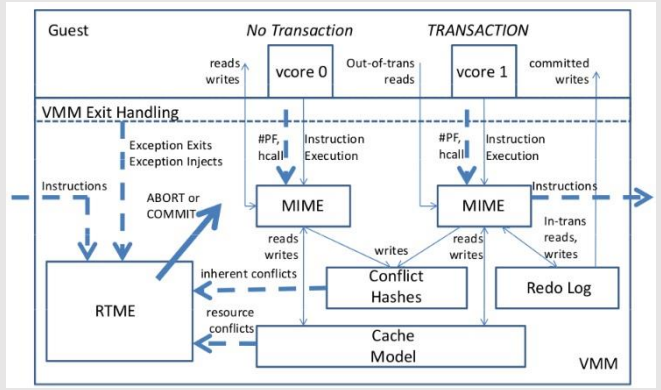
Memory and Instruction Meta-Engine

- Per-core mechanism
- Single-steps through instruction memory accesses
- Driven by shadow page table (sPT) #PF exceptions and hypercalls

Redo Log

- Entries are kept in the form:

{vcore, sequencenum,rip,addr,size,value,type}



System – We have written the RTME and MIME as additions to the Palacios VMM, although our techniques could be implemented in other VMMs. High-level system design:

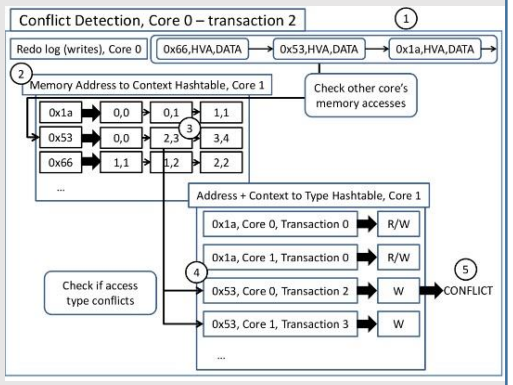
- XBEGIN causes #UD, RTME catches and sets TM mode
- MIME flushes sPT to begin catching memory accesses
- MIME replaces next instruction with hypercall
- MIME maps in pages as needed, using staging page to capture writes made during transaction
- Hypercall indicates end of instruction, old instruction restored, MIME begins new instruction
- XEND, XABORT, and other conditions make RMTE end transaction

For more details, please refer to Technical Report TR NU_EECS_13_03. The code for our system will be incorporated into the publically available Palacios development branch.

System SLOC
~1,300

Conflict Detection

(1) At the end of an instruction in a transactional block, core 0 walks over its redo log of writes. (2) Core 0 checks if any conflicting memory accesses have been made by looking at every other core's address context hash. In the figure, core 0 is checking for conflicting accesses to memory address 0x53. (3) core 0 has found an entry for 0x53 in core 1's address context hash, and walks the list of contexts during which core 1 accessed 0x53. Core 0's current transaction number is 2, so the entry made during context {2,3} is a potential conflict. (4) core 0 checks the entry for address 0x53 in core 1's access type hash to identify the kind of access made. (5) Having found that core 1 wrote to address 0x53 when core 0 was in transaction 2, a conflict is detected, and core 0 must abort its transaction.



Evaluation

In order to evaluate the correctness of our system, we created a small suite of test applications. The suite tested the effect of using XABORT during the middle of a transaction (with writes to memory and without), writing to memory locations, reading from memory locations, writing register values to memory, and threaded accesses to memory (to independent as well as overlapping memory locations). For the threaded tests, each thread was pinned to a different core, and memory accesses were made in a loop, in order to cover all possible orderings of transactions from each core.

Emulation Method	Slowdown vs. Native
RTME/MIME	~1,500x
Intel SDE 5.31.0	~90,000x

~60x faster

Garbage Collection

At the end of any transaction on any core, we start the garbage collection process for the recording structures on that core. By comparing the global context tags for address context entries, we are able to mark entries as garbage, and remove those entries as well as the corresponding entries in the access type hash.

Acknowledgements: This project is made possible by support from the United States National Science Foundation (NSF) via grant CNS-0709168, and the Department of Energy (DOE) via grant DE-SC0005343.