

# Big Idea?

## Virtualizing Transactional Memory

Ravi Rajwar

Microarchitecture Research Lab  
Intel Corporation  
ravi.rajwar@intel.com

Maurice Herlihy<sup>1</sup>

Computer Science Department  
Brown University  
mph@cs.brown.edu

Konrad Lai

Microarchitecture Research Lab  
Intel Corporation  
konrad.lai@intel.com

CSE 240B

Dean Tullsen

CSE 240B

Dean Tullsen

## What Problems are they trying to solve?

CSE 240B

Dean Tullsen

## Where does transactional state reside?

- What, then, has to happen on a transactional memory operation? (assume cache miss)
- How do they make this fast? (two mechanisms)

CSE 240B

Dean Tullsen

# What's in the?

- XSW
- XADT
- XF

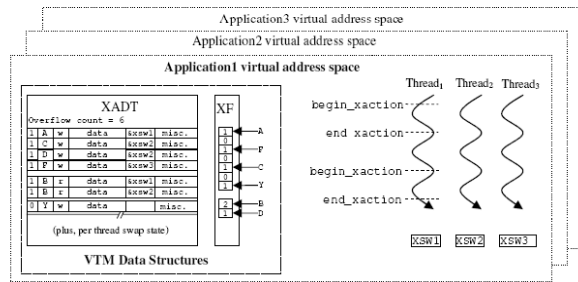


Figure 1 VTM software data structures. A conceptual snapshot of the address space is shown. Threads execute series of transactions. XADT records overflow information, and any swap information. XF summarizes the XADT like a bloom filter. For example, XF has "Y" marked, but "Y" is invalid in XADT.

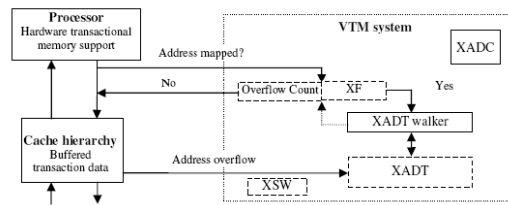


Figure 2 VTM overview. Each processor has its own VTM system machinery. The software-resident XADT and XF data structures are shown with dashed boxes, and the hardware structures are shown with solid boxes. The VTM machinery operates on the XF and XADT using cacheable operations. The XADC caches remapping translation information for overflowed blocks.

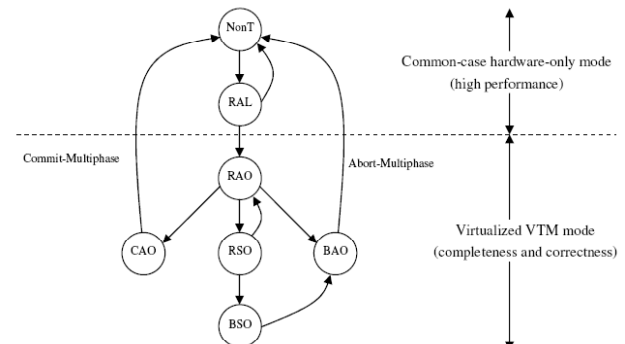


Figure 3 VTM state transition diagram. NonT: Not executing a transaction, R: running, C: committing, B: aborting, A: actively executing, S: swapped out, L: all local hardware, O: overflowed state.

# What happens on a context switch?

# What happens on a commit?

- How is it made atomic?

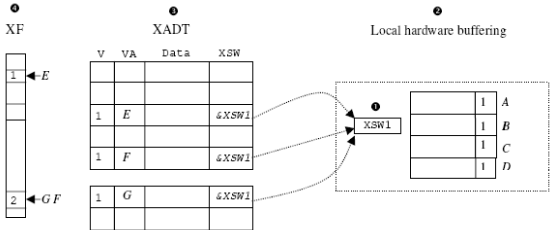


Figure 4 Commit sequence for virtualized transactions in VTm. Here, locations G and F map to the same XF entry.

# What happens on an abort?

# What are nested transactions?

- What is flattening?

# Big Idea?

## Architectural Support for Software Transactional Memory

Bratin Saha, Ali-Reza Adl-Tabatabai, Quinn Jacobson  
Microprocessor Technology Lab, Intel Corporation  
bratin.saha, ali-reza.adl-tabatabai, quinn.a.jacobson@intel.com

CSE 240B

Dean Tullsen

CSE 240B

Dean Tullsen

# Why STM rather than HTM?

CSE 240B

Dean Tullsen

# Basic Implementation Idea?

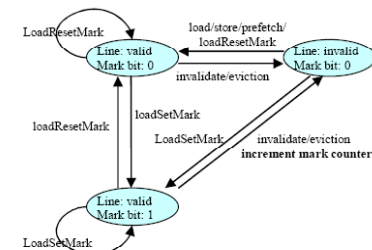


Figure 1: Cache line transitions

CSE 240B

Dean Tullsen

## In STM, what needs to happen on a

- Read
- Store
- Commit
- Abort
- What is “eager version management”?

CSE 240B

Dean Tullsen

## Where is the most performance lost on a STM?

CSE 240B

Dean Tullsen

```
loadTestMark eax, [rec] /* check 1st access */
jnae done
loadSetMark eax, [rec]
test eax, #versionmask /*is a version no.*/
js contentionOrRecursion
mov ecx, [txndesc + rdsetlog] /*get log ptr*/
test ecx, #overflowmask
jz overflow
add ecx, 8 /* inc log ptr */
mov [txndesc + rdsetlog], ecx
mov [ecx - 8], rec /* logging */
mov [ecx - 4], eax /* logging */
done:
```

Figure 5: HASTM object-based read barrier

CSE 240B

Dean Tullsen

```
validate() {
    markCount = readMarkCounter();
    resetMarkAll();
    if (markCount == 0) /*no snoop or eviction*/
        return;
    /* perform full read set validation */
    for <txnrec,ver> in transaction's read set
        if (*txnrec != ver)
            abort();
}
```

Figure 6: HASTM validation

CSE 240B

Dean Tullsen

## What is?

- Aggressive-mode HASTM

CSE 240B

Dean Tullsen

## Results

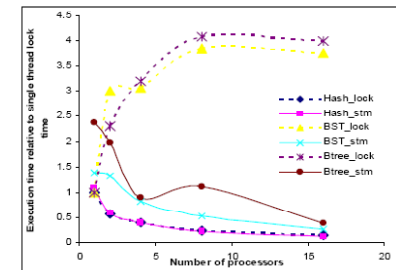


Figure 11: STM (solid lines) vrs lock (dashed line) on TM workloads

CSE 240B

Dean Tullsen

## The problem

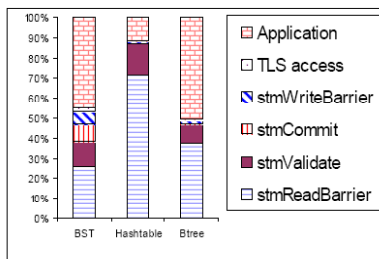


Figure 12: STM execution time breakdown

CSE 240B

Dean Tullsen

## Sequential execution overhead

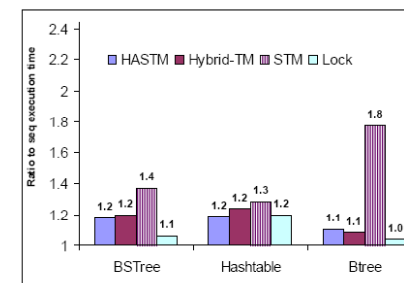


Figure 16: Relative execution time for TM schemes

CSE 240B

Dean Tullsen

# Application Results

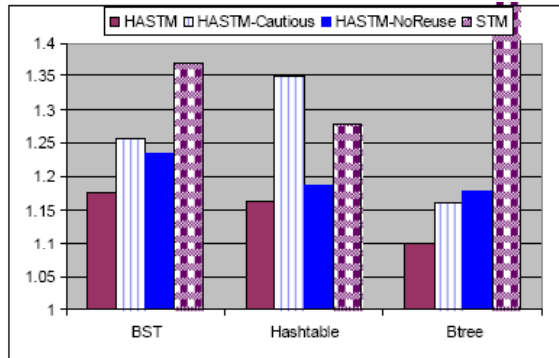


Figure 17: Performance breakdown for HASTM

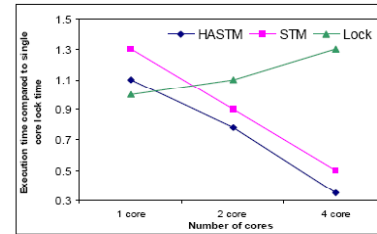


Figure 18: Multi-core scaling for BST

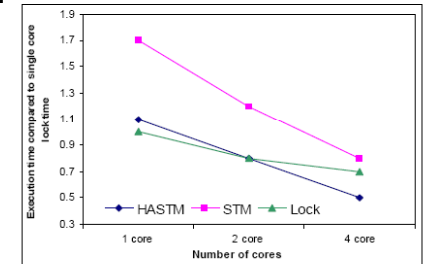


Figure 19: Multi-core scaling for Btree

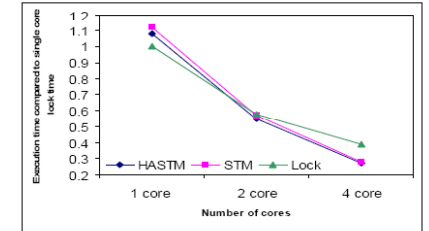


Figure 20: Multi-core scaling for hash table