Virtualizing Transactional Memory

Rajwar, R., Herlihy, M., and Lai, K. 2005

presented by VasilyVolkov, 04/30/08

Motivation

- Transactional Memory is good
  - Never deadlocks
  - Makes concurrent programming easier
- But requires programmer to be aware of
  - Transaction buffer size
  - Scheduling quanta
  - Process migration
- Too low-level issues to be mainstream

Goals

- Transparently hide resource exhaustion
  - In space (cache overflows)
  - In time (scheduling and clock interrupts)
  - *functionality*
- Fast common case
  - Keep virtualization off the critical path
  - *performance*
- Sketch of solution
  - keep data in virtual memory if evicted from local buffer

Overview of Data Structures

- XSW: Transaction Status Word
  - Ultimate authority on transaction's status
  - One per thread
    - to survive context switch / interrupt
    - Nested transactions are flattened
  - Keeps current transaction state:
    - Running/committing/aborted
    - active/swapped out
    - cached/overflowed
    - No transaction

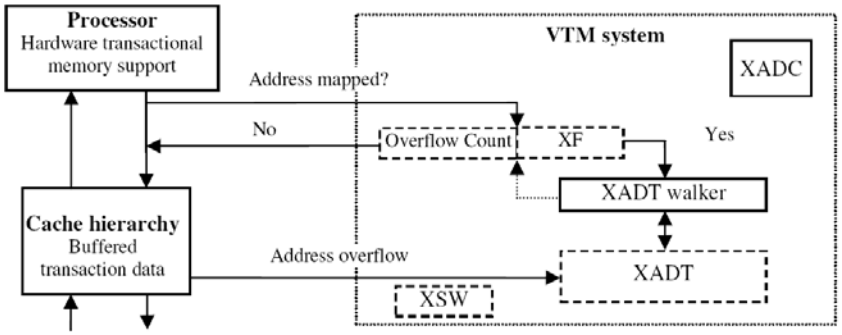
Overview of Data Structures

- XADT: Transaction Address Data Table
  - One per process/address space
  - Is in virtual memory
  - Handles overflow
  - Keeps evicted transaction cache lines:
    - Clean and tentative value
    - Data's virtual address
    - Pointer transaction's status word (TSW)
    - Same transaction's entries are linked for faster cleaning
    - Multiple entries per block if accessed by multiple threads
  - Keeps number of overflowed data blocks
    - To quickly check if it is empty

Overview of Data Structures

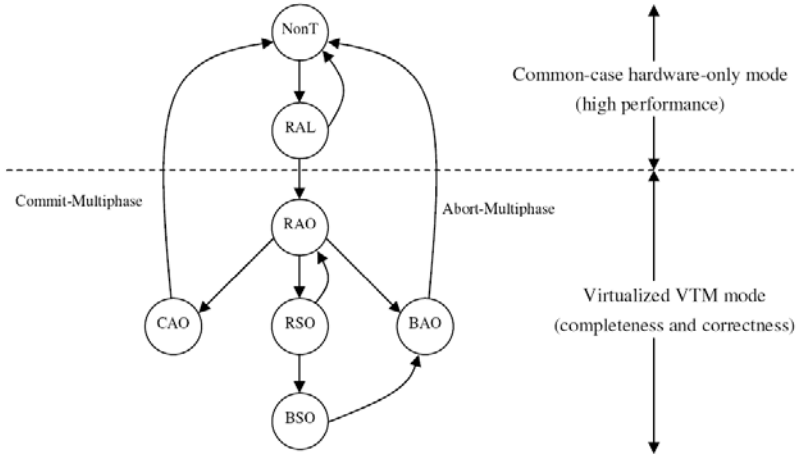
- XF: XADT Filter
- Does an address conflict with XADT?
  - Too slow to walk through XADT every time
  - Use Bloom filter to check quickly if it is not there
    - If Bloom says "may be there" – walk through
  - Use counting Bloom filter to enable removal
  - Estimated to consume up to 10MB of virtual memory
    - Optimizations are possible

Overall Structure of VTM



- Access to XF and XADT is cached
- Dashed: software structures
- Solid: hardware structures

Transaction State Transition



R/B/C = running/aborted/committing
A/S = active/swapped-out
L/O = local/overflowed

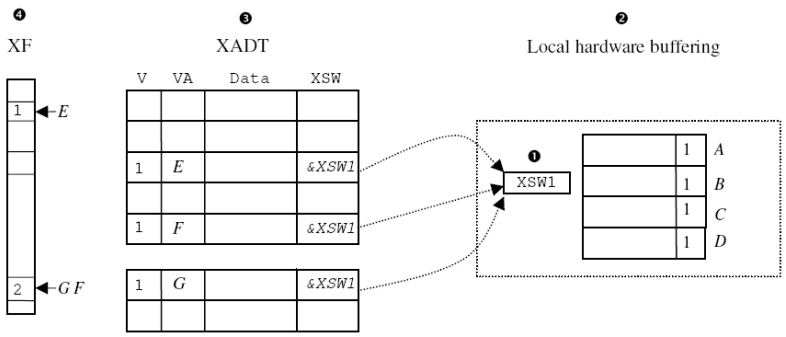
Synchronization w/ non-transactions

- Non-transactional accesses?
  - must not threaten transactions' atomicity!
- Can be done by checking XF and XADT
  - Slows down
- Enough to serialize with transactional accesses
- Non-transactions can't read transaction buffer
  - Can't read value that is later aborted
- Non-transaction CAN write transactional data
  - "poison" data in cache when data goes to XADT
  - Abort if conflict with other process

Context Switch / Page Faults

- Flush data to XADT when thread is suspended
  - Both clean and cached states
- Repopulate on demand when rescheduled
- Check that stored clean values matches current memory values
  - Abort otherwise – conflict with non-transactional access
- Do same on page faults
  - Don't necessarily need to abort on page fault!

Committing and Aborting



1. Update XSW
2. Update local transaction buffer
3. Update XADT
4. Update XF

Can be safely interrupted!

Open Challenges

- Virtual address aliasing
  - 2 processes address same physical data via different virtual addresses
- Effect of system calls within a transaction is not defined
  - OS should have its own XADT?
- Exception handling?
- True nested transactions?