2008
YEAR

PRESENTATION

The University of Auckland | New Zealand

Computer Science 703

Advance Computer Architecture

2008 Semester 1

Lecture Notes for 7May08

Virtualizing Transactional Memory

James Goodman

Department of Computer Science

Test

- Tuesday, 13May, in-class (room 279)
- Coverage: through STM
- Open book, notes
  - Calculators allowed (but not needed)
  - No communication devices

Lecture Time Change!

- There will be a special review session 3-4 on Monday, 12th May.
- After next week, no lectures at 3pm on most Tuesdays/Thursdays due to conflict.
- There will be additional lectures on *Mondays*, **in room 561 (5th floor common room)** including the 19th & 26th of May but **not** the 2nd of June.

Criteria VTMs must meet to integrate into existing systems

- Virtualization must ensure the performance of common-case hardware-only transactional mode is unaffected.
- Conflict detection between active transactions and transactions with overflowed state should be efficient, should not impede unrelated transactions
- Committing and aborting a transaction should not delay transactions that do not conflict
- Context switches and page faults may impede transaction progress, but should not prevent transactions from eventually committing.
- Nontransactional operations may abort transactions but should not compromise any transaction's consistency (strong isolation).
- This virtualization should be transparent to application programmers, much in the way virtual memory management is.

Virtual Transactional Memory

"… a combined *hardware/software* system architecture that allows the programmer to obtain the benefits of transactional memory without having to provide explicit mechanisms to deal with those rare instances in which transactions encounter resource or scheduling limitations. The underlying VTM mechanism transparently hides resource exhaustion both in space (cache overflows) and time (scheduling and clock interrupts). When a transaction overflows its buffers, VTM remaps evicted entries to new locations in virtual memory. When a transaction exhausts its scheduling quantum (or is interrupted), VTM saves its state in virtual memory so that the transaction can be resumed later.

"Virtualization provides essential functionality, but should have an insignificant effect on performance."

Assumptions

- No programming model, semantics, or software policies
- Simple TM programming model
  - multiple s/w threads running in a single shared virtual address space
- Each thread has XSW that is monitored continually by the processor.
- VTM assumes each processor has support for a typical bounded HTM ("best-effort hardware")
  - buffer updates
  - track transactional accesses
  - detect conflicts using h/w mechanisms.
- *Best-effort hardware handles most (nearly all?) cases*

Two Modes of operation

- Hardware-only fast mode ("best-effort hardware" not specified here) provides transactional execution for common-case transactions that
  - do not exceed hardware resources
  - are not interrupted
- Programmer-transparent software structures and hardware machine collectively support transactions that encounter
  - buffer overflow
  - page faults
  - context switches
  - thread migration

Transactional State

- Locally-cached state resides in processor-local buffers
- Overflowed state reside in data structures in the application's virtual memory

XSW

- Each transaction (associated with a single thread) has a Transaction Status Word (XSW)
- VTM implementation commits or aborts a transaction by atomically updating its XSW in two-step process (for overflowed transactions)
  - *Logical* commit
  - Multi-step *physical* commit

Transaction Address Data Table (XADT)

- Keeps track of transactional state that has overflowed from processors to memory.
- Common to all transactions sharing an address space.
- Invoked in two ways
  - a running transaction evicts a cache line
  - an entire transaction may be swapped out, evicting all transactional cache lines.

Interaction of XADT

- On every transactional cache miss, must check XADT
- Mechanisms to minimize overhead
  1. Global bit indicating if XADT is empty, checked by hardware (claimed to be normally empty)
  2. XADT filter detects most cases where line is *not* present in XADT, checked by hardware
  3. Software invoked to check for conflict

XADT Specification

- Like virtual memory, where page size is architecture-specific, cache line size is a system parameter.
- Accesses similar to virtual memory, walking equivalent of page table (without TLB)

Invoking XADT

- On Transactional Cache Overflow…
  - Cache overflow triggers system-supported copy of cache line to XADT
- On Context Switch…
  - Saving transactional buffers to XADT is accomplished as part of context switch
- Overflow state maintained per process, not per processor
  - Can isolate runaway process
  - Easier to detect conflicts if only one process
  - Easier to support debuggers, profiling libraries, etc.
- Tracking done through virtual addresses
  - Processes with transactions in progress can be swapped out

XSW's Three "Dimensions"

1. Transaction State
   - Running (R)
   - Committed, not yet fully visible (C)
   - Aborted (B)
2. Process state
   - Actively executing (A)
   - Swapped out (S)
3. Execution State
   - Cache entirely locally (L)
   - (Partially) Overflowed (O)

Only valid states: RAL, RAO, RSO, BAO, BSO, CAO

XADT: details

- Records an overflow count that is globally visible
- Information stored for each entry
  - Status bits marking
    1. whether the entry is valid
    2. whether a transaction read or write the address
  - Other miscellaneous details
    - Virtual address of the data block overflowed
    - Data field for buffering updates
    - Point to the overflowing transaction's status word (XSW)
    - Content of data, even if only read
    - Other information relevant to transaction (conflict & scheduling priorities, links to other entries, temporary register state

XADT Filter (XF)

Bloom filter provides two operations

- add(x) inserts x into the set
- member(x) queries whether x is in the set
  - May include occasional false positive

How to remove element from Bloom filter?

- Periodically rebuild list
- "Counting" Bloom filter