2008 YEAR

PRESENTATION

The University of Auckland | New Zealand

Computer Science 703

Advance Computer Architecture

2006 Semester 1

Lecture Notes 5

19Mar08

Speculation; Atomic RMW Primitives

James Goodman

Department of Computer Science

How to predict branch decision?

- Brute force: fetch down both paths
- Statically
  - Branch type
    - Special instructions for loop variables
    - Software may predict
  - Forward not take, backward taken
- Dynamically
  - What happened last time(s)?
  - How did we get here?

How Can We Speculate?

A process has *state*, consisting of

- Registers
- Memory
- Distinguish between
  - completing the execution of an instruction
  - changing state

Method 1

- Take a snapshot of state
- On failed speculation, roll back to snapshot
- Can be performed quickly if state is small

Method 2

- Create log of changes of state
- On failure, "unexecute" log
- Recovery is proportional to length of speculation + startup

Important Requirement

Results of speculative execution must never be visible to other threads

- Reading a value may be deferred momentarily, but not indefinitely
- Once value is supplied, it cannot be changed
  - Speculation may have to be aborted

How Do We Speculate?

- Registers are small in number, can be saved as snapshot quickly with hardware support
- Memory is too large to take a snapshot, but
  - Cache is already a snapshot!
  - Save changes in cache and discard on failure

Other Kinds of Prediction: Cache Misses

- Pattern detected (e.g., *stride*)
  - Prefetch data into cache before requested
- Software may predict cache miss
- Thread-level speculation

Thread-level Speculation

Two threads executing "the same" code

- One thread races ahead, but doesn't execute all instructions
  - Just branch instructions and those that affect branch decisions
  - Tests cache on loads/stores. On miss, initiates fetch into cache, but doesn't wait
- Second thread runs behind, hitting in the catch and predicting branches correctly

Other Speculation: Slow Memory Operations

- Store operation hasn't completed yet
  - Load datum
  - Execute speculatively
  - Check before committing

Value Prediction

- Can we predict the value in a register before and instruction writing to it completes?
  - Perhaps, e.g., if it is an index variable
- When we miss in the cache, can we predict the value while we wait?
  - If the cache line fell out of the cache because of capacity, probably not
  - If the cache line was invalidated because another process modified it, possibly so!

False Sharing

- Cache coherence granularity is a cache line
- Two threads are reading and writing disjoint data in the same cache line
- Cache line is "ping-ponging" between caches
- Data is actually in the cache, but marked stale

Value Speculation

Solution to false sharing

- Fetch stale data but initiate cache miss
- Assume that data is correct (i.e., the false sharing is occurring)
  - Take checkpoint
  - Begin speculative execution
- If assumption was incorrect
  - Restore state and execute with correct data

Critical Section

- Discover that lock variable is present in cache, but
  - lock is read-only
  - lock is FREE
- Take checkpoint, but begin executing critical section speculatively as if lock were held
- Abort if
  - lock is invalidated
  - data read during speculation is invalidated by another thread
  - data written during speculation is read by another thread
- When lock release is encountered, commit without acquiring lock!
  - Commit entire critical section simultaneously
  - Lock could have been acquired at the beginning and released at the end
- In zero time, the lock was acquired, the critical section executed, then the lock was released