Portland State University
ECE 588/688

Cache Coherence Protocols

Conditions for Cache Coherence

- **Program Order.** A read by processor P to location A that follows a write by P to A, with no writes to A by another processor in between, should always return the value of A written by P
- **Coherent View of Memory.** A read by processor P1 to location A that follows a write by another processor P2 to location A should return the written value by P2 if:
  - The read and write are sufficiently separated in time
  - No other writes to A by another processor occur between the read and the write
- **Write Serialization.** Writes to the same location are serialized: Two writes to the same location by any two processors are seen in the same order by ***all*** processors

Cache Coherence

- Cache coherence defines behavior of reads and writes to the same memory location
- Cache coherence is mainly a problem for shared, read-write data structures
  - Read only structures can be safely replicated
  - Private read-write structures can have coherence problems if they migrate from one processor to another
- Two main types of cache coherence protocols:
  - Snooping: Caches keep track of the sharing status of all blocks, No centralized state is kept
  - Directory: Sharing status of any block in memory is kept in one location, the directory

Example Codes that May Cause Coherence Problems

- Finite-buffer producer/consumer, paper figure 3
  - Producer generates an item unless buffer is full
  - Consumer removes an item unless buffer is empty
  - Read-write sharing for buffer size and buffer elements
- Solving a linear system of equations, paper figure 4
  - Both array A and vector **b** are read-shared, can be safely replicated
  - Vector **x** is computed every cycle, is read-write shared

Invalidate vs. Update Protocols

- Write-invalidate protocols
  - Guarantees only one writer has a valid copy of a block
  - When a processor wants to write to a cache block, it issues a "Get Exclusive" request to other processors, forcing them to invalidate any copies of the block
  - Subsequent writes from the same processor are done locally in the cache
- Write-update protocols
  - When a processor writes to a block, it sends data to all other processors with valid copies
- Paper figure 5
- Pros and cons?

Write-Once Invalidate Protocol

- States
  - Invalid
  - Valid: Copy is consistent with memory
  - Reserved: Data has been written exactly once, and copy is consistent with memory (the only other copy)
  - Dirty: Data modified more than once, only valid copy
- Copy-back memory update policy: Block is written back to memory when replaced if the block is dirty
- Events:
  - Processor read (P-Read) and Processor write (P-Write)
  - Memory read block (Read-Blk) and write block (Write-Blk)
  - Write-Inv: Invalidate all other copies of block
  - Read-Inv: Read a block and invalidate all other copies
- Paper figure 6

Firefly Update Protocol

- States
  - Valid-exclusive: Only copy, consistent with memory copy
  - Shared: One of many valid copies
  - Dirty: Only valid copy, memory is inconsistent
- Protocol uses copy-back update policy for private blocks
- Protocol uses write-through for shared blocks
- Paper figure 7
- Used in the Firefly multiprocessor workstation from DEC
- Another update protocol: Dragon protocol proposed for the Dragon MP workstation from Xerox
  - Avoids updating memory until a block is replaced

Implementation Issues for Snoopy Coherence Protocols

- Easier to implement compared to directory protocols
  - Directory protocols discussed next time
- Cache Controller: A finite state machine that implements coherence protocol state transition diagram
- Cache Directory: Stores state for each block
- Bus Controller: Implements bus snooping, monitors every bus operation and takes action if needed
- Contention for directory between local and bus requests
- Impact of block size
- Write-through vs. write-back
- Write-allocate vs. write no-allocate

Software Coherence Protocols

- Compiler limits which blocks can be cached
- Types of data accesses
  1. Shared read-only
  2. One writer, multiple readers
  3. One process read/write
  4. Shared read-write
- Trivial solution: All shared read-write blocks are marked as uncacheable (types 2 and 4 above)
- Optimization: some shared read-write variables can be used by one processor for a long time, so may be cached
- Disadvantages vs. hardware protocols?

More Hardware Protocols

- Hardware protocol variations:
  - MSI
  - MESI
  - MOSI
  - MOESI
- Discuss intermediate states

Multi-Level Protocols

- Inclusion/Exclusion policy for multi-level caches:
  - Inclusive caches
  - Exclusive caches
  - Non-inclusive (non-exclusive) caches
- Which caches need to snoop?
- CMP private vs. shared caches
  - Private caches maintain coherence state
  - Shared caches may store state of all L1 caches

Reading & Homework

- Readings for Monday:
  - Daniel Lenoski et al., "The Directory-based Cache Coherence Protocol for the DASH Multiprocessor," ISCA 1990 (Review)
  - James Laudon and Daniel Lenoski, "The SGI Origin: A ccNUMA Highly Scalable Server," ISCA 1997 (Read)
- Homework 2 due today
- Homework 3 out today, due next Wed
- Project proposals due this Monday
  - Send pdf or text document by email