Computer Systems Organization

CSE 521/560M
Lecture 21
Prof. Patrick Crowley

Plan for Today

- Homework 2 and 3
- Homework 4 reminder
- Questions
- Today’s discussion
Directory-based Cache Coherence

- Snooping is not possible on media other than a bus or ring
- Broadcast/multicast is not that easy; performance gets bad, quickly
- Directory-based protocols
  - Data structure records the state of each block and is associated with either
    - Memory modules, or
    - Caches
- The same states and transitions as seen in snooping protocols are associated with each block, only the actions change

Directories in a Distributed Shared-Memory Architecture
Information Associated with Each Block

- Minimally
  - Is the block cached?
  - Is it clean or dirty?
- In order to avoid broadcasts
  - Where is it cached?
    - This is the directory
- Summary: we must record the state of each block and its locations

Full Directory

- Directory: state vector associated with each block
  - For an \( n \) processor system, this is an \( (n+1) \) bit vector
  - Bit 0: clean/dirty
  - Bit \( i \) is set if \( i \)th processor's cache has a copy
  - Protocol is write-invalidate
- Memory overhead
  - For a 64 processor system, 65 bits are needed per block
  - If a block is 64 bytes, overhead = \( 65/(64\times8) > 10\% \)
Basic (Simplified) Protocol

- Details & Definitions
  - Address space *statically partitioned* across nodes (i.e., an address resolves to one and only one directory)
  - Home node: the node that contains the directory
  - Remote node: any other node
- Cache $i$ has a read miss on a clean block
  - Add entry in directory (i.e., set $i$th bit)
- Cache $i$ has a read miss on a dirty block
  - If dirty block is in home node, change its encoding to clean; Add entry in directory (i.e., set $i$th bit); send data
  - If dirty block is not in home node, get it from where it is cached, change entries in directories (at the requestor and the owner) and send the data

Basic Protocol (cont’d)

- Write miss on a clean block
  - Send a series of invalidations, one for each of the caches containing a clean copy; then use the read steps to get data
- What happens when a cache block is replaced?
  - If dirty, it is, of course, written back
  - If clean, nothing really to do. But other directories will not know this, and will still require acknowledgements on any invalidations for the replaced block.
  - Acknowledgements are used in these protocols to ensure correctness when ordered delivery is not guaranteed by the interconnect
Cache Block Transitions

Directory State Transitions
Parallelism Implies Synchronization

- Locking
  - Critical sections
  - Mutual exclusion
- Barriers
  - Provide process synchronization
- Based on primitive: Read-Modify-Write
  - Basic principle: Atomic exchange
  - Test-and-set (0 indicates a free resource; 1 indicates busy)
  - Fetch-and-add (0 indicates a free resource; \( n \) indicates \( n \) users)

Faking Atomicity

- Instead of atomic exchange, have an instruction pair that can be deduced to have operated in an atomic fashion
- Alpha: Load locked (ll) plus Store conditional (sc)
  - Sc detects if the value of the memory location loaded by ll has been modified. If so, it returns 0 otherwise 1.
  - Similar to atomic exchange but does not require read-modify-write
- Implementation
  - Use a special register to store the address of the memory location addressed by ll. On interrupt or invalidation of block corresponding to that address (by another sc), the register is cleared. If on sc the addresses match, the sc succeeds.
Spin Locks

- Repeatedly: try to acquire the lock
- In a cache coherent environment:
  - Can generate many bus transactions (not good)
  - Replace “test-and-set” with “test and test-and-set” (keeping the test local to the cache) or Il+sc (which achieves the same effect)
  - A race condition still exists when the lock is released
  - Contention for a lock causes lock serialization
- To scale up, avoid serialization via
  - Exponential back-off (high penalty for many unsuccessful attempts)
  - Queue locks (keep all contenders in a queue, and release them in order)

Barriers

- All processes have to wait at a synchronization point
- Processes do not progress until all of them have reached the barrier
- Low-performance implementation: use a counter initialized with the number of processes
  - When a process reaches the barrier, it decrements the counter (atomically) and busy waits
  - When the counter is zero, all processes are allowed to progress (logical broadcast)
- Better alternative: use fetch-and-add
Assignment

• Readings
  – For Monday
    • H&P: 6.5, 6.8
    • Commentary: Weak Ordering, A New Definition
  – For Wednesday
    • None