Scalable Cache Coherence

Instructor: Josep Torrellas
CS533
Outline

• Review: Chapter on Cache Coherence from any of the two textbooks


The Cache Coherence Problem

- Caches are critical to modern high-speed processors
- Multiple copies of a block can easily get inconsistent
  - processor writes. I/O writes,..

![Diagram of cache coherence problem]
Cache Coherence Solutions

• Software based vs hardware based
• Software-based:
  – Compiler based or with run-time system support
  – With or without hardware assist
  – Tough problem because perfect information is needed in the presence of memory aliasing and explicit parallelism
• Focus on hardware based solutions as they are more common
Hardware Solutions

- The schemes can be classified based on:
  - Snoopy schemes vs. Directory schemes
  - Write through vs. Write-back (ownership-based) protocols
  - Update vs. Invalidation protocols
  - Dirty-sharing vs. No-dirty-sharing protocols
Snoopy Cache Coherence Schemes

• A distributed cache coherence scheme based on the notion of a snoop
  – watches all activity on a global bus
  – or is informed about such activity by some global broadcast mechanism.
• Most commonly used method in commercial multiprocessors
Write-Back/Ownership Schemes

• When a single cache has ownership of a block, processor writes do not result in bus writes thus conserving bandwidth.
• Most bus-based multiprocessors nowadays use such schemes.
• Many variants of ownership-based protocols exist
Write Through Schemes

• All processor writes result in:
  – update of local cache and a global bus write that:
    • updates main memory
    • invalidates/updates all other caches with that item
• Not used because too much traffic
Invalidation vs. Update Strategies

1. Invalidation: On a write, all other caches with a copy are invalidated
2. Update: On a write, all other caches with a copy are updated
• Invalidation is bad when:
  – producer and (one or more) consumers of data.
• Update is bad when:
  – multiple writes by one PE before data is read by another PE.
  – Junk data accumulates in large caches (e.g. process migration).

• Invalidation schemes are used
Base Invalidation Protocol (MSI)
Illinois Scheme (MESI)

- States: I, VE (valid-exclusive), VS (valid-shared), D (dirty)
- The cache knows if it has an valid-exclusive (VE) copy.
- In VE state no invalidation traffic on write-hits
- Advantage:
  - Closely approximates traffic on a uniprocessor for sequential programs
- Disadvantage:
  - More complex
Illinois Scheme (MESI)

Invalid → Shared

Bus invalidation

P-read [someone has it]

Shared → Valid

Bus-read

P-read

P-read [no one else has it]

Valid → Dirty

Bus invalidation

P-read [someone has it]

Dirty → Invalid

Bus Write Miss

P-write

Bus Write Miss

P-read

P-write

Bus-read

P-read

P-read [no one else has it]

Copyright Josep Torrellas 2012
DEC Firefly Scheme

- Classification: Write-back, update, no-dirty-sharing.
- States:
  - VE (valid exclusive): only copy and clean
  - VS (valid shared): shared-clean copy.
    - Write hits result in updates to memory and other caches and entry remains in this state
  - D(dirty): dirty exclusive (only copy)
- Used special “shared line” on bus to detect sharing status of cache line
- Supports producer-consumer model well
- What about sequential processes migrating between CPU’s?
DEC Firefly Scheme

- **Valid**
  - P-Read
  - Bus Read/Write
  - Bus write-miss
  - P-Write and not SL

- **Exclusive**
  - P-Write and not SL

- **Shared**
  - P-Read and SL
  - Bus Write miss
  - P-Write M and SL
  - P-Write and SL

- **Dirty**
  - P-Write miss
  - Bus-write miss
  - Bus Read
  - [update MM]

- **P-write Miss** and not SL

Copyright Josep Torrellas 2012
Directory Based Cache Coherence

Key idea: keep track in a global directory (in main memory) of which processors are caching a location and the state.

- Motivation: Snoopy schemes do not scale because they rely on broadcast
- Directory based schemes allow scaling
  - They avoid broadcasts by keeping track of all PEs caching a memory block, and then using point-to-point messages to maintain coherence
  - They allow the flexibility to use any scalable point-to-point network
Basic Scheme (Censier and Feautrier)

- Assume K processors
- With each cache-block in memory: K presence bits and 1 dirty bit
Read Miss

Read from main-memory by PE_i
- If dirty bit is off then \{read from main memory; turn p[i] ON;\}
- If dirty bit is ON then \{recall line from dirty PE (cache state to shared); update memory; turn dirty-bit OFF; turn p[i] ON; supply recalled data to PE_i;\}
Write Miss

If dirty-bit OFF then:
{send invalidations to all PE’s caching that block and clear their P[k] bits; turn dirty bit ON; turn P[i] ON; supply data to PE_i } 

If dirty bit ON then:
{recall the data from owner PE which invalidates itself; update memory; clear bit of previous owner; turn bit PE[I] on; (dirty bit ON all the time); forward data to PE_i}
Write Hit to Non-Owned Data

Write- hit to data valid (not owned) in cache:
{access memory-directory; send invalidations to all PE’s
  caching block; clear their P[k] bits; turn dirty bit ON; turn PE[i]
  ON; supply data to PE_i}
Key Issues

• Scaling of memory and directory bandwidth
  – Cannot have main memory or directory memory centralized
  – Need a distributed cache coherence protocol

• As shown, directory memory requirements do not scale well
  – Reason is that the number of presence bits needed grows as the number of PEs.
  – Also: the larger the main memory is, the larger the directory
Directory Organizations

• Memory-based schemes (DASH) vs Cache-based schemes (SCI)

• Cache-based schemes (or linked-list based)
  – Singly linked
  – Doubly-linked (SCI)

• Memory-based schemes (or pointer-based)
  – Full map (Dir-N) vs Partial-map schemes (Dir-i-B, Dir-i-CV-r,...)
  – Dense (DASH) vs Sparse directory schemes
Pointer-Based Coherence Schemes

- The Full Bit Vector Scheme
- Limited Pointer Schemes
- Sparse Directories (Caching)
- LimitLess (Software Assistance)
The Full Bit Vector Scheme

• One bit of directory memory per main-memory block per PE
• Memory requirements are $P \times (P \times M/B)$, where $P$ is the number of PE, $M$ is main memory per PE, and $B$ is cache block size (not counting the dirty bit)
• Invalidation traffic is best
• One way to reduce the overhead is to increase $B$
  – Can result in false sharing and increased coherence traffic
• Overhead may be ok for modest-scale mps
  – Example: 256 PE organized as 64 4-PE clusters with 64-byte cache blocks ---> 12% memory overhead
Limited Pointer Schemes

• Since data is expected to be in only a few caches at any one time, a limited number of pointers per directory entry should suffice
• Overflow strategy: what to do when the number of sharers exceeds the number of pointers?
• Many different schemes based on different overflow strategies
Some Examples

- Dir-i-B
  - Beyond i-pointers, set the inval-broadcast bit ON
  - Storage needed is: \( i \times \log(P) \times PM/B \) (in addition to inval-broadcast bit)
  - Expected to do well since widely shared data is not written often

- Dir-i-NB
  - When sharers exceed i, invalidate one of the existing sharers
  - Significant degradation expected for widely-shared mostly-read data

- Dir-i-CV-r
  - When sharers exceed i, use bits allocated to i pointers as a coarse resolution vector (each bit points to multiple PE)
  - Always results in less coherence traffic than Dir-i-B
  - Example: use Dir-3-CV-4 for 64 processors
Performance of Directories

- Figure 10 in Gupta et al paper
- Figure 7 in Gupta et al paper
LimitLess Directories

• Limit number of pointers

• On overflow:
  – Memory module interrupts the local processor
  – Processor emulates the full-map directory for block
LimitLess Directories Require...

- Rapid trap handler: trap code executes within 5-10 cycles from trap initiation)
- Software has complete access to coherence controller
- Interface to the network that allows the processor to launch and intercept coherence protocol packets