Cache Coherence

Todd C. Mowry
CS 740
November 10, 1998

Topics

• The Cache Coherence Problem
• Snoopy Protocols
• Directory Protocols
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, ...

```
<table>
<thead>
<tr>
<th>P</th>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>A=7</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>A=5</td>
</tr>
</tbody>
</table>

Memory

1

<table>
<thead>
<tr>
<th>P</th>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>A=5</td>
<td></td>
</tr>
</tbody>
</table>

2
```
Cache Coherence Solutions

Software Based:
- often used in clusters of workstations or PCs (e.g., “Treadmarks”)
- extend virtual memory system to perform more work on page faults
  - send messages to remote machines if necessary

Hardware Based:
- two most common variations:
  - “snoopy” schemes
    » rely on broadcast to observe all coherence traffic
    » well suited for buses and small-scale systems
    » example: SGI Challenge
  - directory schemes
    » uses centralized information to avoid broadcast
    » scales well to large numbers of processors
    » example: SGI Origin 2000
Shared Caches

- Processors share a single cache, essentially punting the problem.
- Useful for very small machines. E.g., DPC in the Encore, Alliant FX/8.
  - Problems are limited cache bandwidth and cache interference
  - Benefits are fine-grain sharing and prefetch effects
Snoopy Cache Coherence Schemes

- A distributed cache coherence scheme based on the notion of a snoop that 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.

- Examples: Encore Multimax, Sequent Symmetry, SGI Challenge, SUN Galaxy, ...
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

Examples:
- early Sequent and Encore machines.

Advantage:
- simple to implement

Disadvantages:
- Since ~15% of references are writes, this scheme consumes tremendous bus bandwidth. Thus only a few processors can be supported.
## 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 use such schemes these days.

- Many variants of ownership-based protocols exist:
  - Goodman's write-once scheme
  - Berkeley ownership scheme
  - Firefly update protocol
  - ...

Goodman's Write-Once Scheme

One of the first write-back schemes proposed

Classification: Write-back, invalidation-based

States:
- I: invalid
- V: valid $\implies$ data is clean and possibly in "V" state in multiple PEs
- R: reserved $\implies$ owned by this cache, but main memory is up-to-date
- D: dirty $\implies$ owned by this cache, and main memory is stale
- Cache sees transactions from two sides: (i) processor and (ii) bus.

- Terminology:
  - prm, prh, pwm, pwh: processor read miss/hit, write miss/hit
  - gbr, gbri, gbw, gbi: generate bus read, read-with-inval, bus write, inval
  - br, bri, bw, bi: bus read, read-with-inval, write, inval observed
Illinois Scheme (J. Patel)

States:
- I, VE (valid-exclusive), VS (valid-shared), D (dirty)

Two features:
- The cache knows if it has a valid-exclusive (VE) copy. In VE state, no invalidation traffic on write-hits.
- If some cache has a copy, cache-cache transfer is used.

Advantages:
- Closely approximates traffic on uniprocessor for sequential pgms
- In large cluster-based machines, cuts down latency (e.g., DASH)

Disadvantages:
- Complexity of mechanism that determines exclusiveness
- Memory needs to wait before sharing status is determined
DEC Firefly Scheme

Classification:
- Write-back, update

States:
- VE (valid exclusive): only copy and clean
- VS (valid shared): shared-clean copy. Write-hits result in updates to 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

Advantage:
- Supports producer-consumer model well

Disadvantage:
- What about sequential processes migrating between CPUs?
Invalidation vs. Update Strategies

Retention strategy: When to drop block from cache
   • 1. Exclusive writer (inval-based): Write causes others to drop.
   • 2. Pack rat (update-based): Block dropped only on conflict.

Exclusive writer is bad when:
   • single producer and many consumers of data (e.g., bound in TSP).

Pack rat is bad when:
   • multiple writes by one PE before data is read by another PE (e.g., supernode-to-column update in panel cholesky).
   • junk data accumulates in large caches (e.g., process migration).

Overall, invalidation schemes are more popular as the default.
Hierarchical Cache Coherence

- Hierarchies arise in different ways:
  (a) A processor with an on-chip and external cache
      (single cache hierarchy)
  (b) Large scale multiprocessor using a hierarchy of buses (multi-cache hierarchy)
Single Cache Hierarchies

- **Inclusion property**: Everything in L1 cache is also present in L2 cache.
  - L2 must also be owner of block if L1 has the block dirty
  - Snoop of L2 takes responsibility for recalling or invalidating data due to remote requests
  - It often helps if the block size in L1 is smaller or the same size as that in L2 cache
Hierarchical Snoopy Cache Coherence

• Simplest way to build large-scale cache-coherent MPs is to use a hierarchy of buses and use snoopy coherence at each level.

• Two possible ways to build such a machine:
  (a) All main memory at the global (B2) bus
  (b) Main memory distributed among the clusters
Hierarchies with Global Memory

• First-level caches:
  • Highest performance SRAM caches.
  • B1 follows standard snoopy protocol (e.g., the Goodman protocol).

• Second-level caches:
  • Much larger than L1 caches (set assoc). **Must maintain inclusion.**
  • L2 cache acts as filter for B1-bus and L1-caches.
  • L2 cache can be DRAM based, since fewer references get to it.
Hierarchies w/ Global Mem (Cont)

Advantages:

- Misses to main memory just require single traversal to the root of the hierarchy.
- Placement of shared data is not an issue.

Disadvantages:

- Misses to local data structures (e.g., stack) also have to traverse the hierarchy, resulting in higher traffic and latency.
- Memory at the global bus must be highly interleaved. Otherwise bandwidth to it will not scale.
Cluster Based Hierarchies

**Key idea:** Main memory is distributed among clusters.

- reduces global bus traffic (local data & suitably placed shared data)
- reduces latency (less contention and local accesses are faster)
- example machine: Encore Gigamax

- L2 cache can be replaced by a tag-only router-coherence switch.
Cache Coherence in Gigamax

Router-Coherence switch must know about:

- Local Mp words in remote caches and their state (clean/dirty)
- Remote Mp words in local caches and their state (clean/dirty)
- A write to a local-bus is passed to global-bus if:
  - reference belongs to remote Mp
  - belongs to local Mp but is present in some remote cache
- A read to a local-bus is passed to the global-bus if:
  - reference belongs to remote Mp (and not in cluster cache)
  - belongs to local Mp and is dirty in some remote cache
- A write on global-bus is passed to the local-bus if:
  - reference belongs to local Mp
  - data belongs to remote Mp, but the block is dirty in local cache
- ...

Many race conditions are possible

- e.g., a write-back going out as a request is coming in
Hierarchies: Summary

Advantages:
- Conceptually simple to build (apply snooping recursively)
- Can get merging and combining of requests in hardware

Disadvantages:
- Physical hierarchies do not provide enough bisection bandwidth (the root becomes a bottleneck, e.g., 2-d, 3-d grid problems)
- Latencies often larger than in direct networks
Directory Based Cache Coherence
Motivation for Directory Schemes

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 & Feautrier)

- Assume "k" processors.
- With each cache-block in memory: k presence-bits, and 1 dirty-bit
- With each cache-block in cache: 1 valid bit, and 1 dirty (owner) bit

- 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 to main memory:
  - If dirty-bit OFF then { supply data to PE-i; send invalidations to all PEs
    caching that block; turn dirty-bit ON; turn P[i] ON; ... }
  - ...
Key Issues

Scaling of memory and directory bandwidth
- Can not 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
- In reality, there are many ways to get around this problem
  - limited pointer schemes of many flavors
The Stanford DASH Architecture

- Nodes connected by scalable interconnect
- Partitioned shared memory
- Processing nodes are themselves multiprocessors
- Distributed directory-based cache coherence
Directory Protocol Examples

- Read of remote-dirty data

1. Read Request to Home

2. Forward Read Request to Dirty

3. Sharing Writeback to Home

3. Read Reply to Local

- Forwarding strategy minimizes latency and serialization
Key Issues

Scaling of memory and directory bandwidth
- Can not 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
- ==> How many bits or pointers are really needed?
Cache Invalidation Patterns

• Hypothesis: On a write to a shared location, with high probability only a small number of caches need to be invalidated.

• If the above were not true, directory schemes would offer little advantage over snoopy schemes.
# Invalidation Pattern Summary

<table>
<thead>
<tr>
<th>Pattern Description</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code and read-only objects (e.g., distance matrix in TSP)</td>
<td>no problems as rarely written</td>
</tr>
<tr>
<td>Migratory objects (e.g., particles in MP3D)</td>
<td>even as # of PEs scale, only 1-2 invalidations</td>
</tr>
<tr>
<td>Mostly-read objects (e.g., bound in TSP)</td>
<td>invalidations are large but infrequent, so little impact on performance</td>
</tr>
<tr>
<td>Frequently read/written objects (e.g., task queue data structures)</td>
<td>invalidations usually remain small, though frequent</td>
</tr>
</tbody>
</table>
| Synchronization objects | low-contention locks result in small invalidations  
high-contention locks need special support (SW trees, queueing locks) |
Directory Organizations

Memory-based schemes (DASH) vs. cache-based schemes (SCI)

Cache-based schemes:
  • singly-linked (Thapar) vs. doubly-linked schemes (SCI)

Memory-based schemes:
  • Full-map (Dir-N) vs. partial-map schemes (Dir-i-B, Dir-i-CV-r, ...)
  • Dense (DASH) vs. sparse directory schemes (DASH-2)
Cache-based Linked-list Schemes

Keep track of PEs caching a block by linking cache entries together
  • First proposed by Tom Knight for "Aurora" machine in 1987

Scalable Coherent Interface (SCI) is the most developed protocol
  • uses doubly-linked list for chaining cache entries together
Cache-Based Protocols: Summary

Advantages:
- Directory memory needed scales with number of PEs
- They have addressed all forward progress issues

Disadvantages:
- Requires directory memory to be built from SRAM (same as cache)
- To perform invalidations on write, need to serially traverse caches of sharing PEs (long latency and complex)
- Cache replacements are complex as both forward and backward pointers need to be updated
- In base protocol, read to clean data requires 4 messages (first to memory and then to the head-cache) as compared to 2 messages in other protocols. (Slower and more complex)
Memory-based Coherence Schemes

- The Full Bit Vector Scheme
- Limited Pointer Schemes
- Sparse Directories
- ...

...
The Full Bit Vector Scheme

• One bit of directory memory per main-mem block per PE
• Memory requirements are \[ P \cdot (P \cdot M / B) \], where \( P \) is # of PEs, \( M \) is main memory per PE, and \( B \) is cache-block size.
• Invalidation traffic is best
• One way to reduce overhead is to increase \( B \)
  • Can result in false-sharing and increased coherence traffic
• Overhead not too large for medium-scale multiprocessors.
  • Example: 256 PEs organized as sixty four 4-PE clusters
    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 # of pointers per directory entry should suffice.

Overflow Strategy: What to do when # of sharers exceeds # of pointers

Many different schemes based on differing overflow strategies
Some Examples

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

**DIR-i-NB:**
- When sharers exceed "i", invalidate one of 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 PEs)
- Always results in less coherence traffic than Dir-i-B

**Limitless directories:**
- Handle overflow using software traps
Sparse Directories

Since total # of cache blocks in machine is much less than total # of memory blocks, most directory entries are idle most of the time.

Example:
- 256 Kbyte cache, 16 Mbyte memory per PE ==> >98% idle

Sparse directories reduce memory requirements by:
- using single directory entry for multiple memory blocks
- dir-entry can be freed by invalidating cached copies of a block
- main problem is the potential for excessive dir-entry conflicts
- conflicts can be reduced by using associative sparse directories
FLASH Directory Structure

Use a dynamic pointer scheme (Simoni)

- dense array with single pointer per memory block, plus next ptr
- pointers for other sharers are allocated out of free pool
- with replacements, memory usage is proportional to cache in machine
- pointer management in FLASH is handled by a fully programmable but specialized processor
Directory-Based Coherence: Summary

Directories offer the potential for scalable cache coherence

- no broadcasts
- arbitrary network topology
- tolerable hardware overheads