Lecture 12:

Directory-Based Cache Coherence

Parallel Computer Architecture and Programming
CMU 15-418, Spring 2013
Today: what you should know

- What limits the scalability of snooping-based approaches to cache coherence?
- How does a directory-based scheme avoid these problems?
- How can the storage overhead of the directory be reduced? (and at what cost?)
Implementing cache coherence

The snooping cache coherence protocols from the past two lectures relied on broadcasting coherence information to all processors over the chip interconnect.

Every time a cache miss occurred, the triggering cache communicated with all other caches!

We discussed what information was communicated and what actions were taken to implement the coherence protocol.

We did not discuss how to implement broadcast. (one example is a shared bus)
Recall non-uniform memory access (NUMA) shared memory systems (e.g., Blacklight)

Idea: locating regions of memory near the processors increases scalability: higher aggregate BW and reduced latency (especially when there is locality in the application)

But... efficiency of NUMA system does little good if the coherence protocol can't also be scaled!

Consider the problem: processor accesses nearby memory (good...), but still must broadcast to all other processors it is doing so (bad...).

Some terminology:

- cc-NUMA = “cache-coherent, non-uniform memory access”
- Distributed shared memory system (DSM): cache coherent, shared address space, but architecture implemented by physically distributed memories
One possible solution: hierarchical snooping

Use snooping coherence at each level

Another example: with memory localized with the groups of processors, rather than centralized
One possible solution: hierarchical snooping

Use snooping coherence at each level

Advantages
- Relatively simple to build (already have to deal with similar issues due to multi-level caches)

Disadvantages
- The root becomes a bottleneck (low “bisection bandwidth”)
- Larger latencies than direct communication
- Does not apply to more general network topologies (meshes, cubes)
Scalable cache coherence using directories

- Snooping schemes use broadcast of coherence messages to determine the state of a line in the other caches

- Alternative idea: avoid broadcast by storing information about the status of the line in one place: a “directory”
  - The directory entry for a cache line contains information about the state of the cache line in all caches.
  - Caches look up information from the directory as necessary
  - Cache coherence is maintained by point-to-point messages between the caches (not on broadcast mechanisms)
A very simple directory

- One directory entry per cache line of memory
- One cache line of memory
- Dirty bit: indicates block is dirty in one of the processors’ caches
- $P$ presence bits: indicate whether processor $P$ has line in its cache

Processor

Local Cache

Directory

Memory

Scalable Interconnect
A partitioned directory

Directory partition is co-located with memory it describes

- “Home node” of a line: node with memory holding the corresponding data for the line
  - Example: node 0 is the home node of the orange line, node 1 is the home node of the blue line

- “Requesting node”: node containing processor requesting line
Example 1: read miss to clean line

Read from main memory by processor 0 of the blue line: line is not dirty

- Read miss message sent to home node of the requested block
- Home directory checks entry for block
Example 1: read miss to clean line
Read from main memory by processor 0 of the blue line: line is not dirty

- Read miss message sent to home node of the requested block
- Home directory checks entry for block
  - If dirty bit for block is OFF, respond with contents from memory, set presence[0] to true
Example 2: read miss to dirty line

Read from main memory by processor 0 of the blue line: line is dirty (contents in P2’s cache)

- If dirty bit is ON, then data must be sourced by another processor
- Home node must tell requesting node where to find data
  - Responds with message providing identity of line owner ("get it from P2")
Example 2: read miss to dirty line

Read from main memory by processor 0 of the blue line: line is dirty (contents in P2’s cache)

1. If dirty bit is ON, then data must be sourced by another processor
2. Home node responds with message providing identity of line owner
3. Requesting node requests data from owner
4. Owner changes state in cache to SHARED (read only), responds to requesting node
Example 2: read miss to dirty line

Read from main memory by processor 0 of the blue line: line is dirty (contents in P2’s cache)

1. If dirty bit is ON, then data must be sourced by another processor
2. Home node responds with message providing identity of line owner
3. Requesting node requests data from owner
4. Owner responds to requesting node, changes state in cache to SHARED (read only)
5. Owner also responds to home node, home clears dirty and updates presence bits
Example 3: write miss

Write to memory by processor 0: line is clean, but resident in P1’s and P2’s caches

1. Request: write miss msg
Example 3: write miss

Write to memory by processor 0: line is clean, but resident in P1’s and P2’s caches

1. Request: write miss msg
2. Response: sharer ids + data
Example 3: write miss

Write to memory by processor 0: line is clean, but resident in P1’s and P2’s caches

1. Request: write miss msg
2. Response: sharer ids + data
3. Request: invalidate (2 msgs)
Example 3: write miss

Write to memory by processor 0: line is clean, but resident in P1’s and P2’s caches.

1. Request: write miss msg
2. Response: sharer ids + data
3. Request: invalidate (2 msgs)
   4a. Response: ack from P1
   4b. Response: ack from P2

After receiving both invalidation acks, P0 can perform write.
Advantage of directories

- On reads, directory tells requesting node exactly where to get the line from
  - Either from home node (if the line is clean)
  - Or from the owning node (if the line is dirty)
  - Either way, retrieving data involves point-to-point communication

- On writes, the advantage of directories depends on the number of sharers
  - In the limit, if all caches are sharing data, all caches must be communicated with (just like broadcast)
Cache invalidation patterns

64 processor system

Graphs plot histogram of number of sharers of a line at the time of a write

In general only a few processors share the line (only a few processors must be told of writes)

Note shown: the number of sharers increases slowly with P (good!)
In general, only a few sharers during a write

- **Access patterns**
  - Mostly-read objects: lots of sharers but writes are infrequent, so minimal impact on performance (e.g., root node in Barnes-Hut)
  - Migratory objects: very few sharers, count does not scale with number of processors
  - Frequently read/written objects: frequent invalidations, but few of them because sharer count cannot build up between invalidations (e.g., shared task queue)
  - Low-contention locks: no problem, infrequent invalidations. (high-contention locks do present a challenge)

- **Implication 1:** directories are useful for limiting coherence traffic
  - Don’t need a broadcast mechanism to “tell everyone”

- **Implication 2:** suggests ways to optimize directory implementation (reduce storage overhead)
Full bit vector directory representation

- Recall: one presence bit per node

- Storage overhead proportion to $P \times M$
  - $P =$ number of nodes
  - $M =$ number of blocks in memory

- Scales poorly with $P$
  - Assume 64 byte cache line size
  - 64 nodes ($P=64) \rightarrow 12\%$ overhead
  - 256 nodes ($P=256) \rightarrow 50\%$ overhead
  - 1024 nodes ($P=1024) \rightarrow 200\%$ overhead
Reducing storage overhead of directory

- **Optimizations on full-bit vector scheme**
  - Increase cache block size (reduce M term)
    - What are possible problems with this approach? (consider graphs from last lecture)
  - Place multiple processors in a “node” (reduce P term)
    - Need one directory bit per node, not bit per processor
    - Hierarchical: use snooping protocol among processors in a node

- **We will now discuss two alternative schemes**
  - Limited pointer schemes (reduce P)
  - Sparse directories (reduce M)
Limited pointer schemes

Since data is expected to only be in a few caches at once, a limited number of pointers per directory entry should be sufficient (don’t need information about all nodes)

Example: 1024 processor system

Full bit vector scheme needs 1024 bits per line

Instead, can store ~100 pointers to nodes holding the line ($\log_2(1024)=10$ bits per pointer)

In practice, our workload evaluation says we can get by with far less than this
Managing overflow in limited pointer schemes

Many possible approaches

- If broadcast mechanism exists
  - When more than max number of sharers, revert to broadcast

- No broadcast
  - Do not allow more than a max number of sharers
  - On overflow, newest sharer replaces an existing one (invalidate the old sharer)

- Coarse vector
  - Change representation so that each bit corresponds to K nodes
  - On write, invalidate all nodes a bit corresponds to

- Dynamic pointers
  - Hardware maintains a pool of pointers (free list)
  - Manages allocation of pointers to directory lines
Optimizing for the common case

Limited pointer schemes are a great example of smartly understanding and optimizing for the common case:

1. Workload driven observation: in general the number of cache line sharers is low
2. Make the common case simple and fast: array of pointers for first N sharers
3. Uncommon case is still handled correctly, just with a slower, more complicated mechanism (the program still works!)
4. Extra expense is tolerable, since it happens infrequently
Limited pointer schemes: summary

- Limited pointer scheme reduced directory overhead due to large P

- What about reducing M?
Limiting size of directory: sparse directories

- Key observation: the majority of memory is NOT resident in cache. Coherence protocol only needs sharing information for lines that are in cache
  - So most directory entries are “idle” most of the time
  - 1 MB cache, 1 GB memory per node → 99.9% of directory entries are idle
Sparse directories

Directory at home node maintains pointer to only one node caching line (not a list of sharers)

Pointer to next node stored in the cache line

On read miss: add requesting node to head of list
On write miss: propagate invalidations along list
On evict: need to patch up list (linked list removal)
Sparse directories: scaling properties

Good:
- Low memory storage overhead (one pointer per line)
- Storage proportional to cache size (and list stored in SRAM)
- Traffic on write proportional to number of sharers

Bad:
- **Latency** of write proportional to number of sharers (invalidation of lines is serial)
- Higher implementation complexity
Recall: write miss in full bit vector scheme

Write to memory by processor 0: line is clean, but resident in P1’s and P2’s caches

Original bit-vector scheme sends same number of invalidation messages as sparse directory approach, but invalidation messages can be sent to all processors in parallel.
Optimizing directory-based coherence

- Reducing storage overhead of directory data structure
  - Limited pointer schemes
  - Sparse directories

- Reducing number of messages sent to implement coherence protocol
Recall: read miss to dirty block

Read from main memory by processor 0 of the blue line: line is dirty (contained in P2’s cache)

1. Request: read miss msg
Recall: read miss to dirty block

Read from main memory by processor 0 of the blue line: line is dirty (contained in P2’s cache)
(Note: figure below shows final state of system after operation is complete)

Five network transactions in total
Four of the transactions are on the “critical path” (transactions 4 and 5 can be done in parallel)
- **Critical path:** sequence of dependent operations that must occur to complete operation
Intervention forwarding

Read from main memory by processor 0 of the blue line: line is dirty (contained in P2’s cache)

1. Request: read miss msg
Intervention forwarding

Read from main memory by processor 0 of the blue line: line is dirty (contained in P2’s cache)

2. Home node requests data from owner node (processor 2)
3. Owning node responds
**Intervention forwarding**

Read from main memory by processor 0 of the blue line: line is dirty (contained in P2’s cache)

1. Request: read miss msg
2. Request: intervention read
3. Response: data+dir revision
4. Response: data

4. Home node updates directory, and responds to requesting node with data

Four network transactions in total (less traffic)

But all four of the transactions are on the “critical path.” Can we do better?
Request forwarding
Read from main memory by processor 0 of the blue line: line is dirty (contained in P2’s cache)
Request forwarding

Read from main memory by processor 0 of the blue line: line is dirty (contained in P2’s cache)

1. Request: read miss msg
2. Request: send data to requestor
Request forwarding

Read from main memory by processor 0 of the blue line: line is dirty (contained in P2's cache)

Four network transactions in total
Only three of the transactions are on the critical path (transactions 3 and 4 can be done in parallel)
Summary: directory-based coherence

- Primary observation: broadcast doesn’t scale, but luckily we don’t need to broadcast to ensure coherence because often the number of caches containing a line is small.

- Instead of snooping, just store the list of sharers in a “directory” and check the list as necessary.

- One challenge: reducing overhead of directory storage
  - Use hierarchies of processors or larger line sizes
  - Limited pointer schemes: exploit fact the most processors not sharing line
  - Sparse directory schemes: exploit fact that most blocks are not in cache

- Another challenge: reducing the number of messages sent (traffic) and critical path (latency) of message chains needed to implement coherence operations.
Directory coherence in Intel Core i7 CPU

- Centralized directory for all lines in the L3 cache (note importance of inclusion property)

- Directory maintains list of L2 caches containing line

- Instead of broadcasting coherence traffic to all L2’s, only send coherence messages to L2’s that contain the line (Core i7 interconnect is a ring, it is not a bus)

- Directory dimensions:
  - $P=4$
  - $M =$ number of L3 cache lines