Cache Coherence in NUMA Machines

- Snooping is not possible on media other than bus/ring
- Broadcast / multicast is not that easy
  - In Multistage Interconnection Networks (MINs), potential for blocking is very large
  - In mesh-like networks, broadcast to every node is very inefficient
- How to enforce cache coherence
  - Having no caches (Tera MTA)
  - By software: disallow/limiting caching of shared variables (Cray 3TD)
  - By hardware: having a data structure (a directory) that records the state of each block

Information Needed for Cache Coherence

- What information should the directory contain
  - At the very least whether a block is cached or not
  - Whether the cache copy – or copies – is clean or dirty
- Where are the copies of the block
  - Directory structure associated with the block in memory
  - Linked list of all copies in the caches, including the one in memory

Full Directory

- Full information associated with each block in memory
- Entry in the directory: state vector associated with the block
  - For an *n* processor system, an (*n+1*) bit vector
  - Bit 0, clean/dirty
  - Bits 1-n: "location" vector ; Bit *i* set if *i*th cache has a copy
  - Protocol is write-invalidate
- Memory overhead:
  - For a 64 processor system, 65 bits / block
  - If a block is 64 bytes, overhead = 65 / (64 * 8) i.e., over 10%
  - This data structure is not scalable (but see later)

Home Node

- Definition
  - Home node: the node that contains the initial value of the block as determined by its physical address
  - Home node contains the directory entry for a block
  - Remote node: any other node
- On a cache miss (read and write), the request for data will be sent to the home node
- If a block has to be evicted from a cache, and it is dirty, its value should be written back in the home node

Basic Protocol – Read Miss on Uncached/clean Block

- Cache *i* has a read miss on an uncached block (state vector full of 0's)
  - The home node responds with the data
  - Add entry in directory (set clean and *i*th bit)
- Cache *i* has a read miss on a clean block (clean bit on; at least one of the other bits on)
  - The home node responds with the data
  - Add entry in directory (set *i*th bit)

Basic Protocol – Read Miss on Dirty Block

- Cache *i* has a read miss on a dirty block
  - If dirty block is in home node, say node *j* (dirty and *j*th bits on) home node:
    - Updates memory (write back from its own cache *j*)
    - Changes the block encoding (dirty -> clean and set *i*th bit);
    - Sends data to cache *i* (1-hop)
  - If dirty block is not in home node but is in cache *k* (dirty and *k*th bits on), home node
    - Asks cache *k* to send the block and updates memory
    - Change entry in directory (dirty -> clean and set *i*th bit);
    - Sends the data (2-hops)

Basic Protocol – Write Miss on Uncached/clean Block

- Cache *i* has a write miss on an uncached block (state vector full of 0's)
  - The home node responds with the data
  - Add entry in directory (set dirty and *i*th bits)
- Cache *i* has a write miss on a clean block (clean bit on; at least one of the other bits on)
  - Home node sends an invalidate message to all caches whose bits are on in the state vector (this is a series of messages)
  - The home node responds with the data
  - Change entry in directory (clean -> dirty and set *i*th bit)
- Note : the memory is not up-to-date

Basic Protocol – Write Miss on Dirty Block

- Cache *i* has a write miss on a dirty block
  - If dirty block is in home node, say node *j* (dirty and *j*th bits on) home node:
    - Updates memory (write back from its own cache *j*)
    - Changes the block encoding (clear *j*th bit and set *i*th bit);
    - Sends data to cache *i* (1-hop)
  - If dirty block is not in home node but is in cache *k* (dirty and *k*th bits on), home node
    - Asks cache *k* to send the block and updates memory
    - Change entry in directory (clear *k*th bit and set *i*th bit);
    - Sends the data (2-hops)

Basic Protocol – Request to Write a Clean Block

- Cache *i* wants to write one of its block which is clean
  - This implies that clean/dirty bits also exist in the cache metadata
  - Perform as in write miss on a clean block except that the memory does not have to send the data

Basic Protocol - Replacing a Block

- What happens when a block is replaced
  - If dirty, it is of course written back and its state becomes a vector of 0's
  - If clean could either "do nothing" but then encoding is wrong leading to possibly unneeded invalidations (and acks) or could send message and modify state vector accordingly (reset corresponding bit)
  - Acks are necessary to ensure correctness mostly if messages can be delivered out of order

The Most Economical (Memory-wise) Protocol

- Recall the minimal number of states needed
  - Not cached anywhere (i.e., valid in home memory)
  - Cached in one or more caches but not modified (clean)
  - Cached in one cache and modified (dirty)
- Simply encode the states (2-bit protocol) and perform broadcast invalidations (expensive because most often the data is not shared by many processors)
- Fourth state to enhance performance, say valid-exclusive:
  - Cached in one cache only and still clean: no need to broadcast invalidations on a request to write a clean block but the cache has to know that it is in v-e state (metadata in the cache)

2-bit Protocol

- Differences with full directory protocol
  - Of course no bit setting in "location" vector
  - On a read miss to uncached block go to state valid-exclusive
  - On "request to write a clean block" from a cache that has the block in valid-exclusive state, if the block is still in valid-exclusive state in the directory, no need to broadcast invalidations
  - On a read miss to a valid-exclusive block, change state to clean
  - On a write miss to clean block and to valid-exclusive block from another cache and read/write miss to dirty block, need to send a broadcast invalidate signal to all processors; in the case of dirty, the one with the copy of the block will send it back along with its ack.

Need for Partial Directories

- Full directory not scalable.
  - Location vector depends on number of processors
  - Might become too much memory overhead
- 2-bit protocol invalidations are costly
- Observation: Sharing is often limited to a small number of processors
  - Instead of full directory, have room for a limited number of processor id's.

Examples of Partial Directories

- Coarse bit-vector
  - Share a "location" bit among 2 or 4 or 8 processors etc.
  - Advantage: scalable since fixed amount of memory/block
- Dynamic pointer (many variations)
  - Directory for a block has 1 bit for local cache, one or more fields for a limited number of other caches, and possibly a pointer to a linked list in memory for overflow.
  - Need to "reclaim" pointers on clean replacements and/or to invalidate blindly if there is overflow
  - Protocols are Dir_iB (i pointers and broadcast) or Dir_iNB (i pointers and No Broadcast, i.e., forced invalidations)

Directories in the Cache -- The SCI Approach

- Copies of blocks residing in various caches are linked via a doubly linked list
  - Doubly linked so that it is easy to insert/delete
- Header in the block's home
  - Insertions "between" home node and new cache
- Economical in memory space
  - Proportional to cache space rather than memory space
- Invalidations can be lengthy (list traversal)

A Caveat about Cache Coherence Protocols

- They are more complex in the details than they look!
- Snoopy protocols
  - Writes are not atomic (first detect write miss and send request on the bus; then get block and write data -- only then should the block become dirty)
  - The cache controller must implement "pending states" for situations which would allow more than one cache to write data in a block, or replace a dirty block, i.e., write in memory
  - Things become more complex for split-transaction buses
  - Things become even more complex for lock-up free caches (but it's manageable)

Subtleties in Directory Protocols

- No transaction is atomic.
- If they were treated as atomic, deadlock could occur
  - Assume block A from home node X is dirty in P1
  - Assume block B from home node Y is dirty in P2
  - P1 reads miss on B and P2 reads miss on A
  - Home node Y generates a "purge" for B in P2 and Home node X generates a "purge" for A in P1
  - Both P1 and P2 wait for their read misses and cannot answer the home node purges hence deadlock.
- So assume non-atomicity of transactions and allow only one in-flight transaction per block (nack any other while one is in progress)

Problems with Buffering

- Directory and cache controllers might have to send/receive many messages at the same time
  - Protocols must take into account finite amount of buffers
  - This leads to possibility of deadlocks
  - This is even more important for 2-bit protocol with lots of broadcasts
  - Solutions involve one or more of the following
    - separate networks for requests and replies so that requests don't block replies which free buffer space
    - each request reserves buffer room for its reply
    - use of nacks and of retries