Computer Science 703

Advance Computer Architecture

2006 Semester 1

Lecture Notes

29Mar06

Distributed Shared Memory

James Goodman

Department of Computer Science

Example of Protocol Choice

- When data is modified and a read miss occurs, two possibilities:
  1. Supply the data as shared, keep a shared copy, and update memory.
  2. Supply the data as modified, invalidate copy, and possibly update memory.
- Which is better?
  - If data is about to be written (migratory sharing), modified copy should be supplied
  - If data is written occasionally, shared by many, shared copy should be supplied

Protocol Choice Example

- Observation: if modified copy is always supplied, shared state will never be achieved, even if data is widely shared!
- Solution: Remember whether data was locally modified or received modified.
  - Locally modified: migratory data—send modified and purge
  - Locally unmodified: likely shared—send shared copy

Limits to Snooping

- Buses have poor electrical properties, cannot be clocked at high speed
- Snooping depends on broadcast, so every cache must observe every operation intended for memory
- Observations
  - Caches don't have to observe data, just address
  - Data take up most of bandwidth
- **Idea 1**: Allocate single bus for address, use separate network to transmit data

Limits to Snooping

- Observation: Different addresses can be sent over different buses
- **Idea 2**: Split address space into pieces and build multiple independent (interleaved) snooping systems
- Observation: all requests must be seen in some order, but limits of physical bus can be overcome by utilizing multiple physical buses
- **Idea 3**: Send all requests to a single point where they are serialized and broadcast on multiple physical buses, then collect the results and return them to the appropriate requestor

Distributed Shared Memory

References

- Hennessy & Patterson, *Computer Architecture: A Quantitative Approach* (3rd Ed.), 2003, Morgan Kaufmann, San Francisco, CA, USA. Section 6.5: Distributed Shared-Memory Architectures, pp. 576-584.
- L.M. Censier & P. Feautrier, "A new solution to coherence problems in multicache systems," *IEEE Transactions on Computers* ***27****(12)*, pp. 1112-1118, Dec. 1978.
- D. Lenoski, J. Laudon, K. Gharachorloo, W.-D. Weber, A. Gupta, J. Hennessy, M. Horowitz, & M.S. Lam, "The Stanford Dash multiprocessor," *IEEE Computer*, **25**(3), pp. 63-79, 1992.
- J. Laudon & D. Lenoski, "The SGI Origin: a ccNUMA highly scalable server," *Proceedings, 24th Annual International Symposium on Computer Architecture*, Denver, CO, USA, pp. 241-251, 1997.

Directory-based Multiprocessing

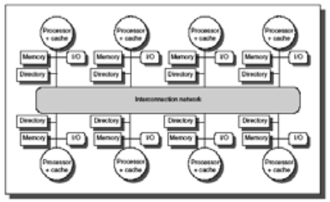

FIGURE 6.27 A directory is added to each node to implement cache coherence in a distributed-memory multiprocessor. Each directory is responsible for tracking the caches that share the memory addresses of the portion of memory in the node. The directory may communicate with the processor and memory over a common bus, as shown, or it may have a separate port to memory, or it may be part of a central node controller through which all intranode and internode communications pass.

Hennessy & Patterson, *Computer Architecture: A Quantitative Approach* (3rd Ed.), p. 578.

Basic Protocol

Similar to three states of MESI protocol

- Don't need to distinguish between M & E

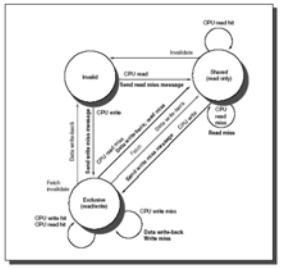

FIGURE 6.29 State transition diagram for an individual cache block in a directory-based system. Requests by the local processor are shown in black and those from the home directory are shown in gray. The states are identical to those in the snooping case, and the transactions are very similar, with explicit invalidate and write-back requests replacing the write misses that were formerly broadcast on the bus. As we did for the snooping controller, we assume that an attempt to write a shared cache block is treated as a miss; in practice, such a transaction can be treated as an ownership request or upgrade request and can deliver ownership without requiring that the cache block be fetched.

Hennessy & Patterson, *Computer Architecture: A Quantitative Approach* (3rd Ed.), p. 581.

Basic Messages

Message type	Source	Destination	Message contents	Function of this message
Read miss	Local cache	Home directory	P, A	Processor P has a read miss at address A; request data and make P a read sharer.
Write miss	Local cache	Home directory	P, A	Processor P has a write miss at address A; — request data and make P the exclusive owner.
Invalidate	Home directory	Remote cache	A	Invalidate a shared copy of data at address A.
Fetch	Home directory	Remote cache	A	Fetch the block at address A and send it to its home directory; change the state of A in the remote cache to shared.
Fetch/invalidate	Home directory	Remote cache	A	Fetch the block at address A and send it to its home directory; invalidate the block in the cache.
Data value reply	Home directory	Local cache	D	Return a data value from the home memory.
Data write back	Remote cache	Home directory	A, D	Write back a data value for address A.

FIGURE 6.28 The possible messages sent among nodes to maintain coherence are shown with the source and destination node, the contents (where P=requesting processor number), A=requested address, and D=data contents), and the function of the message. The first two messages are miss requests sent by the local cache to the home. The third through fifth messages are messages sent to a remote cache by the home when the home needs the data to satisfy a read or write miss request. Data value replies are used to send a value from the home node back to the requesting node. Data value write backs occur for two reasons: when a block is replaced in a cache and must be written back to its home memory, and also in reply to fetch or fetch/invalidate messages from the home. Writing back the data value whenever the block becomes shared simplifies the number of states in the protocol, since any dirty block must be exclusive and any shared block is always available in the home memory.

Hennessy & Patterson, *Computer Architecture: A Quantitative Approach* (3rd Ed.), p. 580.

Cache Line is Uncached

When a block is in the uncached state the copy in memory is the current value, so the only possible requests for that block are

- *Read miss*—The requesting processor is sent the requested data from memory and the requestor is made the only sharing node. The state of the block is made shared.
- *Write miss*—The requesting processor is sent the value and becomes the Sharing node. The block is made exclusive to indicate that the only valid copy is cached. Sharers indicates the identity of the owner.

Hennessy & Patterson, *Computer Architecture: A Quantitative Approach* (3rd Ed.), p. 583.

Cache Line is Shared

When the block is in the shared state the memory value is up-to-date, so the same two requests can occur:

- *Read miss*—The requesting processor is sent the requested data from memory and the requesting processor is added to the sharing set.
- *Write miss*—The requesting processor is sent the value. All processors in the set Sharers are sent invalidate messages, and the Sharers set is to contain the identity of the requesting processor. The state of the block is made exclusive.

Hennessy & Patterson, *Computer Architecture: A Quantitative Approach* (3rd Ed.), p. 583.

Cache Line is Exclusive

When the block is in the exclusive state the current value of the block is held in the cache of the processor identified by the set sharers (the owner), so there are three possible directory requests:

- *Read miss*—The owner processor is sent a data fetch message, which causes the state of the block in the owner's cache to transition to shared and causes the owner to send the data to the directory, where it is written to memory and sent back to the requesting processor. The identity of the requesting processor is added to the set sharers, which still contains the identity of the processor that was the owner (since it still has a readable copy). *Note data could also be sent as modified.*
- *Data write-back*—The owner processor is replacing the block and therefore must write it back. This write-back makes the memory copy up to date (the home directory essentially becomes the owner), the block is now uncached, and the sharer set is empty.
- *Write miss*—The block has a new owner. A message is sent to the old owner causing the cache to invalidate the block and send the value to the directory, from which it is sent to the requesting processor, which becomes the new owner. Sharers is set to the identity of the new owner, and the state of the block remains exclusive.

Hennessy & Patterson, *Computer Architecture: A Quantitative Approach* (3rd Ed.), p. 583.

How to Maintain a Sharing List

- A bit vector requires one bit for each processor that might share
  - Is this scalable?
- A list of variable size—each element stores a processor number.
  - Store a small maximum number, then broadcast
  - Allocate space dynamically and build linked list

When is a Write Completed?

When all valid copies have been made unreadable

- Invalidations may be sent serially
- Generally requires acknowledgement to be sure
- Acknowledgements sent to directory? Requestor?

Races

Message requests can be delayed or delivered out of order

- What happens if a read request arrives while another cache holding a modified cache line is writing it back to memory?
- What happens if two processors attempt to write the same cache line at the same time?

Other Possibilities

- Observation: number of cached lines is limited by total size of (combined) caches
  - Size of all combined sharing lists is small
  - Can "cache" sharing lists
- List of sharers can be distributed among sharing nodes
  - If cache has a shared copy, it must also supply a pointer to another copy
  - A distributed, linked-list is maintained
  - Directory need only keep track of head of list.
  - Implemented in Scalable Coherent Interface (SCI)

Comparing Snooping and DSM

- Snooping is inherently non-scalable
  - Serial requirement limits growth as more processors are added
- DSM can scale (but note problems with sharing list)
  - But a simple cache miss results in at least 3 serial transmissions
  - Write operations can be very slow

Hybrid Models

- DASH used DSM to extend snooping clusters
- Point-to-point links can be much faster
  - Use point-to-point links to broadcast like snooping
  - Use protocol messages to
    - assure proper serialization of operations
    - detect and resolve conflicts