Module 14: "Directory-based Cache Coherence"

Lecture 33: "SCI Protocol"

Directory-based Cache Coherence:

Special Topics

- Sequent NUMA-Q
- SCI protocol
- Directory overhead
- Cache overhead
- Handling read miss
- Handling write miss
- Handling writebacks
- Roll-out protocol
- Snoop interaction
- Protocol processor

[From Chapter 8 of Culler, Singh, Gupta]
[SGI Origin 2000 material taken from Laudon and Lenoski, ISCA 1997]
[GS320 material taken from Gharachorloo et al., ASPLOS 2000]

Previous Next

Sequent NUMA-Q

- Implements the IEEE SCI directory protocol
  - One node is an Intel Pentium Pro quad SMP
  - The IQ-Link board connects to the system bus and implements the directory protocol
    - Also contains a 32 MB 4-way set associative RAC
  - Processors within a node are kept coherent via a MESI snoop-based protocol already implemented in Pentium Pro quad
  - The SCI protocol keeps the RACs coherent across nodes
  - The RAC maintains inclusion with the processor caches

SCI protocol

- Directory structure
  - Home contains the id of the most recently queued sharer or the owner (6 bits)
- Sharing list
  - A sharer contains the id of the next sharer and the previous sharer
  - The last sharer contains the id of home node and previous sharer
  - A circular doubly linked list
- Three major states in directory
  - Home: remotely unowned, but may be in local quad
  - Fresh: same as shared
  - Gone: some node has exclusive ownership; memory stale
- Cache states
  - Processor cache: MESI
  - RAC: 29 stable states and many transient states
    - 7 bits for representing RAC state
    - Two-part naming of RAC state: first part says the location of the block in the list (ONLY, HEAD, TAIL, MID), second part mentions the actual state (modified, exclusive, fresh, copy, …)
    - We will use some of these to understand the basics of SCI (full description available from IEEE standards)
      - HEAD_DIRTY, TAIL_CLEAN, etc
- Three major operations on the list
  - List construction: involves adding a new sharer to the list
  - Rollout: remove a sharer from the list; must synchronize with immediate neighbors
  - Purge/invalidate: head node always has write permission and so it can purge the entire list before writing; naturally, only the head node has the privilege of doing this
- Three classes of protocol
  - Minimal SCI: sharing not allowed
  - Typical SCI (will discuss this): all supports that a normal human being can imagine
  - Full SCI: lot of optimizations including hardware support for synchronization

Previous Next

Directory overhead

- Directory overhead
  - Need 6 bits to maintain the head node id
    - NUMA-Q scales up to 64 nodes
  - Need 2 bits for encoding three states: HOME, FRESH, GONE
  - A system with P nodes, M bytes of memory, and cache block size of B bytes has M/B cache blocks per node
    - 2 + log(P) bits needed for directory entry per cache block
    - Total overhead = (M/B)*(log (P) + O(1))*P
  - O(P*log(P))

Cache overhead

- Extended RAC tags for storing upstream and downstream pointers
  - 2*log(P) per cache block
  - Total increased tag DRAM area is O(P*log(P))

Handling read miss

- Requester on missing the RAC as well as quad snoop sends a read request to home
  - Allocates a block in RAC and marks its state PENDING
  - CASE A: directory is HOME state
    - Change directory state to FRESH
    - Change head pointer to requester id
    - Send reply to requester
    - Requester fills cache block in RAC, forwards it to requesting processor, changes RAC block state to ONLY_FRESH
  - CASE B: directory state is FRESH
    - Home changes head pointer to requester id
    - Sends reply with data read from memory and the old head node id
    - Requester sends a request to the previous head expressing intention to become the new head
    - Old head changes its upstream pointer to point to the requester and the RAC state to MID_VALID or TAIL_VALID; sends an acknowledgment to requester
    - Requester changes its downstream pointer to old head and upstream pointer to home; also changes RAC line state to HEAD_FRESH
    - Observe the strict request-reply nature of the protocol
  - CASE C: directory state is GONE
    - Means head node has an exclusive copy of the cache line
    - Home replies to the requester with the head node id, but does not change the state of the directory
    - Requester sets RAC line state to PENDING and sends a data request to the head node
    - Old head changes RAC line state to TAIL_VALID, sets its upstream pointer to the requester, and sends data to requester
    - Requester sets RAC line state to HEAD_DIRTY, sets its upstream pointer to home and downstream pointer to old head

- Note that directory remains in GONE state and memory is not updated (similar to an M to O transition)

- Handling races
  - Suppose when the requester's (say A) message reaches the old head (say B) the RAC line is in PENDING state
  - SCI doesn't have any pending state in directory or doesn't use NACKs (actually uses, but small in number)
  - B does become the new head (has to because the home has already updated the directory), but inherits the PENDING state from A
  - Any subsequent request will come to B and will become the new pending head
  - Ultimately the PENDING state is resolved along the chain starting from A upstream
  - FIFO nature of the pending list guarantees fairness
  - Also, no problem related to sizing the buffers for holding pending requests (no extra space needed

Previous Next

Handling write miss

- CASE A: requester is in HEAD_DIRTY state already
  - Directory must be in GONE state
  - Only need to invalidate sharers
  - Requester sends an invalidation to the next sharer
  - A sharer upon receiving an invalidation sends a roll-out request to its next sharer (unless TAIL); the receiving node sets its upstream pointer properly and sends a roll-out acknowledgment
  - Eventually roll-out request is acknowledged, the sharer invalidates its RAC line and sends a reply back to head with the id of the next sharer
  - Head moves on to purge the sharer with received id
  - During the entire process requester's RAC line remains in PENDING state
  - Note that home is not at all involved here
- CASE B: requester is in ONLY_DIRTY state
  - No transaction needed
- CASE C: requester is in HEAD_FRESH state
  - Send state change request to home (FRESH to GONE)
  - Once acknowledgment from home is received list purging can be started
  - What if the home is in a state different from FRESH with a different head node?
    - The only case in SCI when a NACK is generated
    - The requester on receiving the NACK changes its state to PENDING and initiates a new write request to home for transitioning to ONLY_DIRTY
- CASE D: requester in MID_FRESH or TAIL_FRESH state
  - First it must roll out from the list and attach itself to the head in HEAD_FRESH state (recall that only the head node can write)
  - This roll-out may require acknowledgments from upstream and downstream neighbors (if MID) or just the upstream neighbor (if TAIL)
  - Follow CASE C
- CASE E: requester not a sharer
  - First get the block in HEAD_DIRTY state
  - Follow CASE A

Previous Next

Handling writebacks

- Requires the evicting node to roll out
  - Same for clean replacements also
  - Dirty eviction (requiring a data transaction to home) can happen only from the head node
    - Requires the head node to roll out
  - Clean eviction can happen from any node in the list
    - Does not require a transaction to home unless its state is ONLY_FRESH or HEAD_FRESH
    - ONLY_FRESH eviction changes directory state from FRESH to HOME (i.e. no sharer)
    - HEAD_FRESH eviction must update the head pointer in directory (directory state remains unchanged)
  - Dirty eviction is completed first before initiating the miss generating the eviction
    - Rationale is low complexity, and RAC eviction is rare

Roll-out protocol

- Some details about the roll-out mechanism
  - CASE A: rolling out from the middle of the list
    - Request-acknowledgment protocol between the victim and its upstream and downstream neighbors
    - If one of the neighbors is in PENDING state it can NACK the roll-out request; the requester must retry
    - Problem arises when two adjacent nodes try to roll out simultaneously (nothing stops both nodes to replace the same cache line at the same time)
      - Both will keep on NACKing each other leading to a livelock
      - To break this cycle the node closer to tail is given priority (how do you know who is closer to tail?)
    - Neighbors may need to change RAC state depending on situation (HEAD_DIRTY to ONLY_DIRTY or HEAD_FRESH to ONLY_FRESH)
  - CASE B: Roll-out from head of the list
    - Neighbor must update RAC state to reflect the fact that it is the new head
    - Home also should be notified about the new head (directory state may not always change)
    - Problem arises when the head change message reaching the home finds a totally new head already registered
      - Means some other node is in the process of attaching itself to the head
      - Home NACKs the roll-out
      - Rolling out node remains in PENDING state and keeps on retrying until the request from the new would-be head arrives
      - At this point the list goes back to stable state and the roll-out can complete

Snoop interaction

- Interesting design problems arise due to limitations of the Pentium Pro quad

- The biggest problem is that the MESI protocol is designed for in-order response (so what?)
- Had to use the deferred response signal for remote requests
  - Lesson learned: for hierarchical protocols bus must be split-transaction with out-of-order response (what happens otherwise?)
- Snoop response is available after four cycles earliest
  - Stall wire may be asserted by any processor unable to meet this four-cycle limit
  - Bus controller samples the stall wire every two cycles
- RAC and directory (for local requests) are also looked up in parallel

Protocol processor

- NUMA-Q runs protocols in microcode
  - The protocol processor is customized with bit-field operations and is a three-stage dual issue pipeline
  - Has dedicated cache for holding recently accessed directory entries and RAC tags
  - Protocol processor also contains three counters for monitoring performance
    - These counters can be programmed through protocol code (i.e. read and written to)

Previous Next