Directory-based Cache Coherence:

Special Topics

- AlphaServer GS320
- Virtual network: Case Studies
- Coherence controller occupancy
- Protocol occupancy
- Directory controllers
- Flexible protocol engine

[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

AlphaServer GS320

- Recall that SGI Origin 2000 eliminates NACKs related to late and early interventions
  - Late interventions are replied by home via writeback forwarding
  - Early interventions are buffered at writer until write is completed
- Origin 2000 still uses NACKs if directory state is busy
- GS320 eliminates all NACKs
  - Simply doesn't have busy states
  - How do you serialize transactions?
- Eliminating PSH: dirty sharing
  - Same as a standard MOESI protocol, but state change in directory is immediate
  - Suppose node P0 is caching a block in M state
  - Node P1 issues a read request to home
  - Home forwards it to P0, changes directory state to clean, and marks P0 as the owner (need an owner field in directory)
  - P0 supplies data to P1 and moves to O state
    - P1 could also become O (is it better or worse?)
  - All subsequent requests are forwarded to P0 by home
  - P0 must serialize them properly
    - Philosophy: keep home free, serialize in the periphery
- Dirty sharing
  - Problem arises if owner evicts the cache block
  - Now home cannot figure out what to do
    - Directory only specifies the sharers and the owner
    - Home does not know exactly which sharers did not get the cache block from P0
    - Home only writes the block back to memory, marks that there is no owner for this block, and sends a writeback acknowledgment to owner
  - Owner in all cases must source the cache block until the writeback is acknowledged
    - Must hold evicted cache blocks in a writeback buffer
  - More problems: what if a request arrives at home before the WB, but reaches owner after the WB ACK?
- Dirty sharing
  - GS320 maintains total order in the network
    - Needed by other optimizations related to invalidation acknowledgments also
  - What if the protocol allows the ownership to move along the sharer chain?
    - New problem: writeback ordering
    - Easy to resolve at home: only accept data from owner marked in the directory entry
    - Always acknowledge writebacks
    - Still need to rely on network order? No, if there are two types of writeback acknowledgments
- Eliminating the PDEX state: write forwarding
  - Same as read case with ownership changing along a chain
- Performance considerations
  - How will a migratory sharing pattern perform on GS320?
  - How will a large-scale producer-consumer pattern perform on GS320?
  - Any special considerations for LL/SC locks?

- Note that lock acquire is essentially a large-scale producer-consumer pattern with the number of consumers decreasing from P-1 to zero

Previous Next

Virtual network: Case Studies

- Each virtual network consists of an NI queue in each direction connected to the corresponding queue or group of queues in the router
- SGI Origin 2000
  - Two virtual networks; uses back-off intervention and invalidation to avoid cycles in the network dependence graph
- Stanford DASH
  - Two virtual networks; in case an incoming request needs space in outgoing request network and outgoing request queue is full, it waits for a pre-defined number of cycles and then if still full, sends a NACK to the requester
- AlphaServer GS320
  - Three virtual networks; longest transaction is 3-hop
- Stanford FLASH
  - Four virtual networks; longest transaction is 4-hop (special case of reply generating a reply)
- Alpha 21364 router
  - 19 virtual channels (essentially queues) in each direction per port: 3 channels per virtual network, six coherence message types, one extra channel forms the seventh virtual network to carry some special coherence control messages (3 channels within a network are used for adaptive routing)

Coherence controller occupancy

- How long does it take to service a message on average?
  - If you imagine the coherence controller as a centralized server in a queuing model, occupancy is just the reciprocal of service rate
  - Occupancy of servicing a message induces a waiting time on the subsequent messages (shows up as a contention component in the total end-to-end latency)
    - Queuing analysis and simulation show that contention grows faster than quadratic in occupancy (Chaudhuri et al, 2003); later empirically confirmed by other researchers that it is likely to be sub-cubic
    - Goal should be to design low-occupancy protocols

Protocol occupancy

- Goal is to design low-occupancy protocol
  - Doesn't mean cannot do smart things
  - A high-occupancy protocol can still perform well if it can reduce the message count accordingly
  - Latency tolerating techniques such as prefetching usually puts more pressure on the coherence controller (why?)
    - Leads to an increased average protocol occupancy
  - Some bad protocol decisions
    - Invalidation acknowledgments at home
    - Replacement hints
    - NACKs
  - Final design is usually influenced by directory organization and coherence controller

microarchitecture

Directory controllers

- Two main designs
  - Hardwired finite state machines (fixed protocol)
  - Software protocol running on embedded protocol processor in memory controller (suited for off-chip memory controllers) or protocol thread in main processor (suited for multi-threaded processors) or protocol core in main processor (suited for multi-core processors)
- Hardwired FSM
  - Low occupancy (all-hardware)
  - Protocol must be simple enough to be able to design and verify in hardware
  - Possible to pipeline various stages of protocol processing
  - Cannot afford late-binding or flexibility in the choice of protocol
  - SGI Origin 2000, MIT Alewife, Stanford DASH

Previous Next

Flexible protocol engine

- Software protocol
  - Executes short sequences of instructions or micro-code known as protocol handlers on a processor
  - Each message type has a separate handler
  - Can make the protocol complicated
  - Allows late-binding of protocol, can choose appropriate protocol, easier verification path
  - Normally higher occupancy than hardwired controllers if controller clock is slow
  - Protocol processor may use separate protocol data and code caches to speed up protocol processing
- Four existing designs
  - Customized coprocessor embedded in memory controller
    - ISA designed to include bit field operations: helpful for directory manipulation (bit clear, bit set, branch on bit clear, branch on bit set, find first set bit, etc.)
    - Processor is normally simple e.g. short pipeline, in-order, no fp unit or mult/div
    - Example: Stanford FLASH, Sun S3.mp, Alpha Piranha CMP, Sequent STiNG, Sequent NUMA-Q
- Four existing designs
  - General purpose processor embedded in memory controller
    - Uses commodity processor cores
    - May be wasteful of resources
    - Normally higher occupancy than customized coprocessor if memory clock is slow
    - Example: Wisconsin Typhoon
- Four existing designs
  - Execute on main processor
    - Interrupt the main processor to execute coherence protocol on cache miss or network message arrival
    - Needs an extremely low overhead interrupt mechanism to be competitive
    - Grahn and Stenstrom (1995)
- Four existing designs
  - Execute on spare hardware thread context of multi-threaded (or hyper-threaded) processors
    - No interrupt overhead
    - Reserve a protocol thread context
    - Application and protocol threads co-exist in the processor (no context switch needed)
    - Chaudhuri and Heinrich (2004)
    - Can’t discuss in detail before talking about SMT/HT
- Possible future design
  - Devote a core to protocol processing in multi-core architectures (Kalamkar, Chaudhuri, and Heinrich, 2007)
  - Increasingly attractive as number of cores increases

Previous Next