Snooping and Distributed Multiprocessor Design

We consider more details about how a bus-based SMP works, and then consider the complications when there is no bus or other central point of communication.

1

Basic Correctness Properties

- Deadlock-free -- no cyclic buffer dependencies
- Livelock-free -- controllers preemptively steal resources from each other without completing
- Starvation-free -- a process does not make progress while others do
- Cache Coherence
- Possibly Sequential Consistency

2

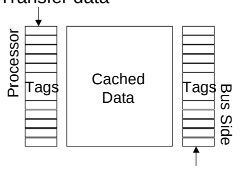
Basic Assumptions of Design

- Single Level Cache
- Transactions on bus atomic
- Cache can stall process to perform multi-action updates -- makes actions look atomic w.r.t. each other

3

Cache Tags and Controller

- Standard bus operations from cache controller
  - Assert Request
  - Wait for bus grant
  - Drive address and command
  - Wait for command to be accepted
  - Transfer data



4

Reporting Snoop Results

- The snoopers must come to some “decision” about bus transactions so memory can know if it’s supposed to deliver data ... when and how?
- Fixed delay counted in clock cycles
  - Snoopers check their tag set -- could be locked because processor is updating
  - Add ability to extend
  - Fixed delay may not be conservative but it works
  - Pentium Pro, HP and Sun processor use this

5

Reporting Snoop Results, II

- Variable delay -- memory assumes the caches will deliver until all caches have said they won’t
  - Allows variable amount of time for snooper to reply, say because it is locked out by processor
  - SGI Challenge uses variable, but with speculative access
- Memory can keep a bit per block indicating whether it is in a cache dirty
  - Doesn’t need snoopers, but uses memory

6

Signalling Their Data

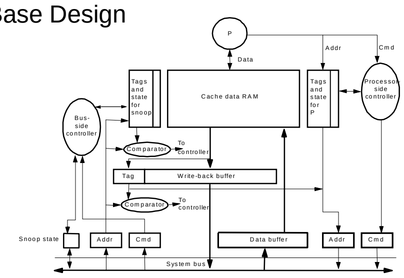
- Three control lines suffice for the protocols we've discussed:
  - Shared -- drive control line if any processor, except the requesting processor, has a copy of the block
  - Modified -- drive control line if the processor has the block in modified state
  - Release -- every controller drives line until it has processed request, then release means others are OK

Note complications if (clean) data can be delivered from multiple caches ... need priority to pick one

Write-backs Affect The Design

- A write-back occurs on a cache miss so two blocks are involved
  - To get processor started fast, move the block to a write-back buffer, and fetch the new block
  - Write-back buffer must snoop bus in case there is a read for a block being written back; if so, cancel write-back and deliver the data

Base Design



Atomicity

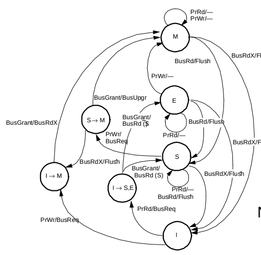
- Even with atomic bus the protocol requires multiple operations by multiple controllers, and multiple requests can be outstanding at once
- P1 wants to perform a BusRdX but cannot get access, meanwhile P2 is performing a BusRd on data P1 has modified

Consider two caches simultaneously issuing write to same block they hold shared

- P1 promotes S-->M and issues upgrade
- P2 does too, but wins arbitration
- P1 downgrades M--> I, but upgrade request till out
- P1's revises to be BusRdX
- Therefore, snoop against requests

Transient State Diagram

Transient states are not part of the state bits in the cache, but are implied by protocol behavior



Note S --->M, I--->M etc.

Serialization

- To speed writes, it may seem smart to let the processor go while the snooper is getting exclusive access to block and possibly filling it
- But other writes might be asserted in this interval trashing coherence if write to same block, or SC if write to any block -- vulnerability
- To be conservative a processor has to be stalled until the BusRdx is complete and write is visible to other processors

But There Is An Optimization

- Relax “completed” to “committed”
- Then, it is sufficient to be asserting exclusive ownership for writes on the bus, since all caches will see that even, the serializing event
- This is sufficient for
  - Coherence
  - Sequential Consistency
- Notice that write-backs are really separate and need not be ordered

Fetch Deadlock, Write Livelock

- Situation: Two controllers have data to service the other’s request but they don’t do so until their request is fulfilled
- Fix: Service requests while waiting for yours
- In invalidation protocol, consider all processors trying to write to one location ... by the time a processor has it in the cache and ready to write, it is invalidated by some other processor
- Fix: Let processor write if it is granted exclusive ownership

Implementing Test&Set

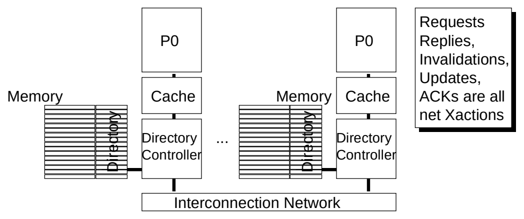
- Two operations: read and write
- Should the lock be cacheable
  - Yes, get locality and spin in cache
  - No, get faster response
- To get atomicity, lock-down the bus between the read and write components
- Sweeter solution: Read the value exclusively, but don’t yield exclusivity until the write is done

Shared Memory Without A Bus

- The bus is a centralized point where writes and reads can be serialized
- How are coherency and sequential consistency achieved without a bus?
- It is possible to broadcast, but this is both expensive and potentially very complicated
- Directory-based cache coherence is solution

Directory

A directory is a data structure giving the state of each cache block in the machine



Preliminaries

- The machines being considered are called distributed shared memory (DSM) class
- The subclass is the CC-NUMA, cache coherent, non-uniform memory access
- On an access-fault by the processor ...
  - Find out information about the state of the cache block in other machines
  - Determine exact location of copies, if necessary
  - Communicate with other controllers to implement the protocol

Terminology

- Home node, node whose main memory has block allocated
- Dirty node, node with modified value
- Owner, node holding valid copy, usually the home or dirty node
- Exclusive node, holds only valid cached copy
- Requesting node, (local) node asking for blk
- Locally allocated / remotely allocated

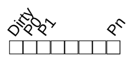
Sample Directory Scheme

- Local node has access fault
- Sends request to home node for directory info
  - Read -- directory tells which node has valid data
    - Data is requested
  - Write -- directory tells nodes with copies
    - Invalidation or update requests are sent
- Acknowledgments are returned
- Processor waits for all ACKs for completion

Notice that many transactions can be “in the air” at once, leading possibly to races

A Directory Entry

- Directory entries do not usually keep cache state
- Use a P-length bit vector to tell in which processors the block is present ... presence bit
- Clean/dirty bit implies exactly 1 presence bit on
- Sufficient?
  - Determine who has a valid copy for read miss
  - Determine who has copies to be invalidated



A Closer Look I (Read)

Postulate 1 processor per node, 1 level cache, local MSI protocol

- On a read access fault at P_x, the local directory controller determines if block is locally/remotely allocated, and if remote finds home
- Controller sends request to home node for blk
- Home controller looks up directory entry of blk
  - Dirty bit OFF, controller finds blk in memory, sends reply, sets x^{th} presence bit ON

A Closer Look II (Read)

- Dirty bit ON -- controller sends reply to P_x of processor with Id, P_y of owner
- P_x controller sends request to owner P_y for data
- Owner P_y controller, sets state to “shared”, forwards data to P_x and sends data to home
- At home, data is updated, dirty bit is turned OFF and the x^{th} presence bit is set ON; notice y^{th} presence bit remains ON

This is essentially the protocol for the LLNL S-1 multicomputer from late ‘70s

A Closer Look I (Write)

- On a write access fault at P_x, the local directory controller determines if block is locally/remotely allocated; if remote finds home
- Controller sends request to home node for blk
- Home controller looks up directory entry of blk
  - Dirty bit OFF, the home has a clean copy
    - Home node sends data to P_x w/ presence vector
    - Home controller clears directory, sets x^{th} bit ON and sets dirty bit ON
    - Px controller sends invalidation requests to all nodes listed in the presence vector

A Closer Look II (Write)

- Px controller awaits ACKs from all those nodes
- Px controller delivers block to cache in dirty state

– Dirty bit is ON

- Home notifies owner P_y of P_x's a write request
- P_y controller invalidates its blk, sends data to P_x
- Home clears y^th presence bit, turns x^th bit ON and dirty bit stays ON

– On writeback, home stores data, clears both presence and dirty bits

When shared is replaced in P_x cache, notifying home is optional -- a replacement hint -- to avoid invalidate

Alternative Directory Schemes

- The "bit vector directory" storage-costly
- Consider improvements to Mblk*P cost
  - – Increase block size, cluster processors
  - – Just keep list of Processor Ids of sharers
    - Need overflow scheme
    - Five slots suffices
  - – Link the shared items together
    - Home keeps the head of list
    - List is doubly-linked
    - New sharer adds self to head of list
    - Obvious protocol suffices, but watch for races

Assessment

- A obvious difference between directory and bus solutions is that for directories, the invalidate request scales as the number of processors that are sharing
- Directories take memory --
  - 1 bit per block per processor + c
  - If a block is B bytes, 8B processors imply 100% overhead to store the directory

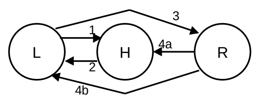
Performance Data

To see how much sharing takes place and how many invalidations must be sent, experiments were run

- Summarizing the data
  - Usually, there are few sharers
  - The mode is 1 other process sharing, ~60
  - The "tail" of the distribution stretches out for some applications
- Remote activity increases as the number of processors
- Larger block sizes increase traffic, 32 is good

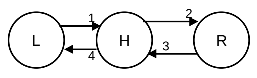
Protocol Optimizations I

- Read request to exclusively held block



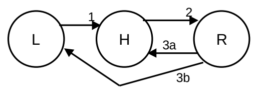
1: Request
2: Response
3: Intervention
4a: Revise
4b: Response

Strict Req/ Response



1: Request
2: Intervention
3: Response
4: Response

Intervention Forwarding

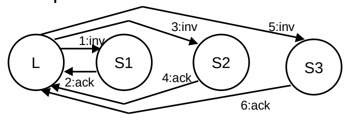


1: Request
2: Intervention
3a: Revision
3b: Response

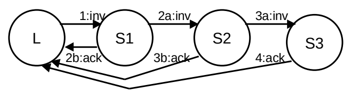
Reply Forwarding

Protocol Optimizations II

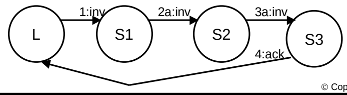
- Improved invalidation



Ack includes next sharer on list



Ack and next invalidate in parallel



Ack comes from last sharer

Higher Level Optimization

- Organizing nodes as SMPs with one coherent memory and one directory controller can improve performance since one processor might fetch data that the next processor wants ... it is already present
- The main liability is that the controller resource, and probably its channel into the network are shared

Serialization

- The bus defines the ordering on writes in SMPs
- For directory systems, memory (home) does
- If home always had the value, FIFO would work

Consider a block in modified state and two nodes request exclusive access in an invalidation protocol. The requests reach home in one order, but they could reach the owner in a different order. Which order prevails?

- Fix: Add a “busy state” indicating a transaction is in flight

Four Solutions To Ensure Serialization

- Buffer At Home -- keep request at home, service in order ... lower concurrency, overflo
- Buffer at requesters with linked list
- NACK and retry -- when directory is busy, just “return to sender”
- Forward to dirty node -- serialize at home for clean, serialize at dirty node otherwise

Origin 2000

- Intellectual descendant of Stanford DASH
- Two processors per node
- Caches use MESI protocol
- Directory has 7 states
  - Stable: unowned, shared, exclusive (cl/dirty in $)
  - Busy states: Processor not ready to handle new requests to that block: read, readex, uncached
  - Poison: used for other purposes
- Directory uses extended bit-vector
- HUB is the interface

Origin-2000 Directory

- The “approximate” bit-vector solution
  - Two processors / node
  - Scaling beyond 64 processors necessary
- Three interpretations are possible
  - Exclusive state: bits are *processor* address
  - Two sizes -- 16-bit and 64-bit vectors
  - Coarse vector -- P/64 nodes are grouped
  - The last two schemes are dynamically selected in large configurations

Specific Choices

- Generally the Origin 2000 follows the protocols discussed with minor variations and optimizations
- The specifics are interesting because they emphasize two points:
  - The basic ideas discussed really apply
  - Many simplifying assumptions must be revisited to get a system built and deployed