Snoop-Based Multiprocessor Design

Module-4

Hemangee K. Kapoor
Department of CSE
IIT Guwahati

Single-Level caches with an Atomic Bus (1)

Module-4
Lecture-2

List of design issues

- Here we have some assumptions but they are physically realistic
- List of preliminary design issues
  1) Design of cache controller and tags. Both processor and bus need to lookup
  2) How and when to present snoop results on the bus
  3) Dealing with write-backs, as they may cause race conditions
  4) Overall set of operations for memory access are not atomic. This can introduce race conditions
  5) Any issues that may arise wrt deadlock, livelock, starvation, serialisation, etc.

(i) Cache controller + Tag design

- Uniprocessor cache
  - Data, tags, state, comparator, controller and bus interface
- Processor cache interaction
  - Compare tags, hit/miss, update state bits, may need block transfer to/from memory
- Cache controller initiates bus operation
  - Assert request for bus; Wait for bus grant; Drive address and command lines; Wait for command to be accepted by relevant device; Transfer data

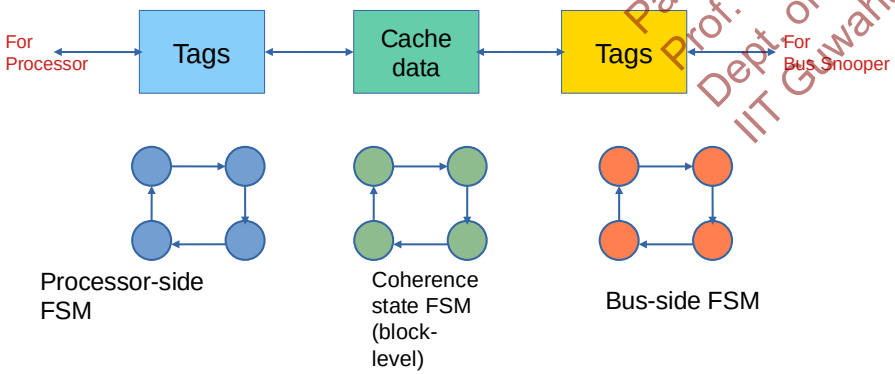
Extend uniprocessor cache

- Cache controller is = FSM
- Sequence of steps in bus transactions = FSM
- Note that coherence protocol is another block-level FSM
- Changes
  - Monitor bus
  - Respond to processor request
- To implement snooping protocol, cache controller has bus-side and processor-side FSM

Duplicate tags

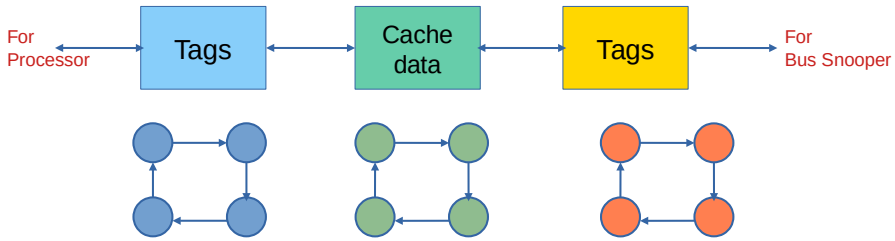
- Processor-side compares tag
- Snoop also compares tags to take appropriate actions
- Cache controller sends Bus transactions in both cases
- Tag comparison needed by both
- Therefore dual-tags (don't duplicate data) or use dual-ported RAM for tags
  - Minimal stall on tag updates. General operation of tag compare is fast

Duplicate tags + FSMs



Cache controller extends

- Cache controller also responds to bus transactions. Has to compare bus address with own address
- If hit = may have to put data on bus
- For update protocol = snoop data from bus and update own cache



(ii) Reporting Snoop Results: When?

- All caches snoop on bus and do tag compare. Collective result of (tag compare) snoop must be sent on bus before transaction can proceed
- This is required as Memory must know if it should give data. i.e. Memory or cache will give
- Keep decision delay to minimum
- Three options:
  - Fixed Delay
  - Variable Delay
  - Immediately

Snoop results in Fixed Delay

- After fixed number of clock cycles after the address appears on bus
- This needs dual-tags to reduce contention with processor
- Still must be conservative, as CPU may lock both tags arrays when updating state (e.g. ‘E’ -> ‘M’)
- Advantage is main memory design not affected
  - Cache-to-cache handshake is simple
- Dis-advantage is extra hardware, Potentially longer latency
- Ex: Pentium Pro, HP Server, Sun Enterprise

Snoop results in Variable Delay

- Memory assumes cache will supply data, until all caches say otherwise
- Less conservative, as we need not assume worst case delay to compute snoop results
- More flexible, more complex (handshakes required between cache <--> memory)
- Memory can however start fetching data. If cache gives data then memory data is not used and memory access stopped
- Ex: SGI Challenge

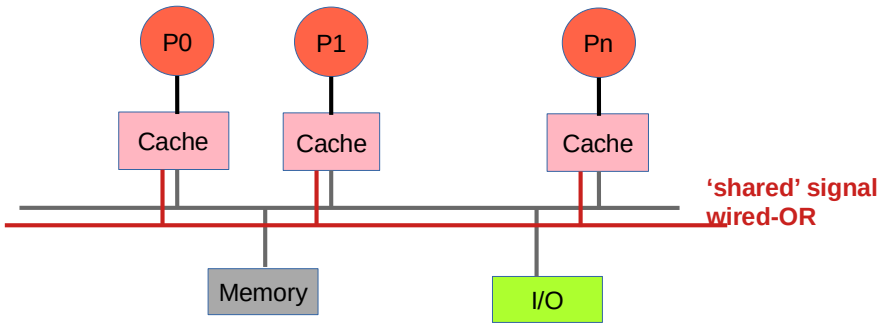
Snoop results Immediately

- Memory maintains 1-bit per block to indicate if it is being modified or not by any cache
- This bit helps memory to decide if memory should send data. Need not wait for snoop results
- Dis-advantage is extra hardware complexity to memory sub-system

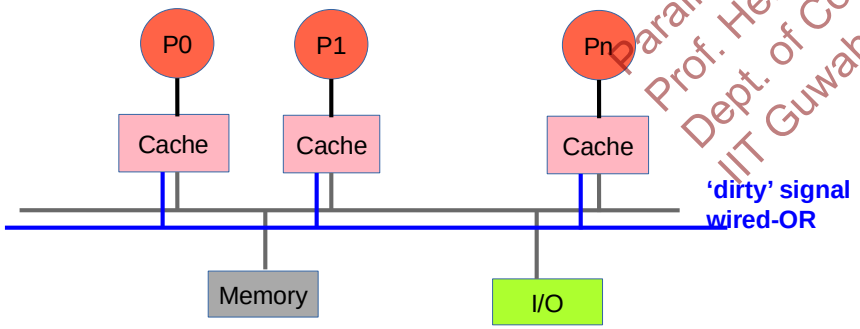
(iii) Reporting Snoop Results: How?

- Collective response from caches must appear on the bus
- Ex: in MESI protocol, we need to know
  - Is block dirty?
  - Should memory respond with data or some cache will provide data
  - Is block shared?
  - This is required to decide if one should load in ‘E’ or ‘S’ state

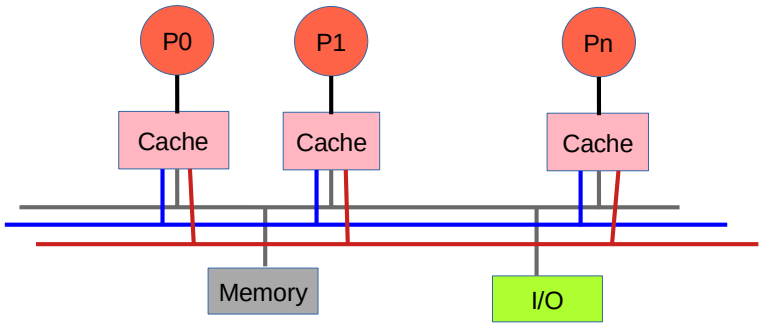
Shared signal



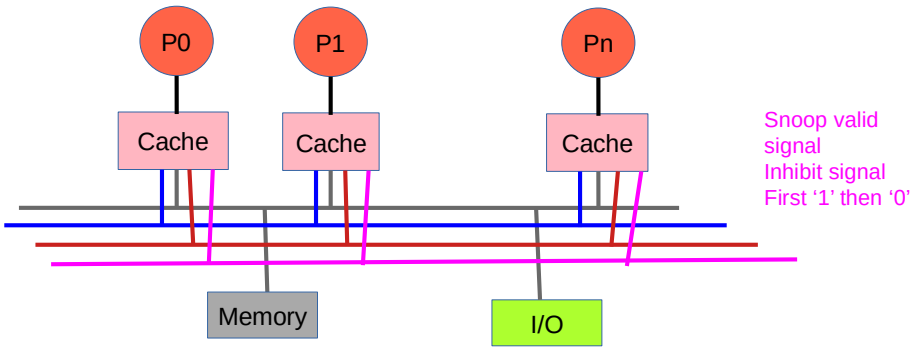
Dirty signal



Shared+ dirty signal



Snoop-valid signal



Wired-OR signals

- Use three wired-OR signals
  1) Shared: asserted if any cache (except Requestor) has a copy
  2) Dirty: asserted if any cache has modified copy. Need not know which, since it will know what to do
  3) Snoop-Valid: this is an inhibit signal. Asserted until all caches have completed their snoop. When de-asserted, Memory and Requestor can examine other two signals

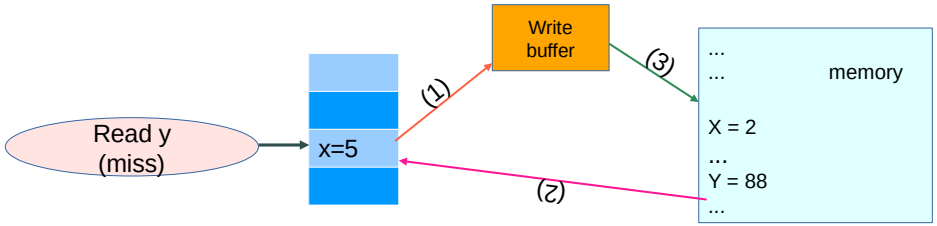
Who provides data?

- Full Illinois-MESI is more complex as
  - Block is provided by cache and not Memory
  - Then priority scheme as to which cache gives block if multiple have it
- Therefore commercial systems avoid cache-to-cache transfer
- Ex: SGI Challenge and Sun Enterprise use cache transfers only for modified data
  - SGI Challenge --> updates Memory when cache gives copy
  - Sun Enterprise --> uses 'O' bit. MOESI and Memory does not update when cache gives

(iii) Dealing with write-backs

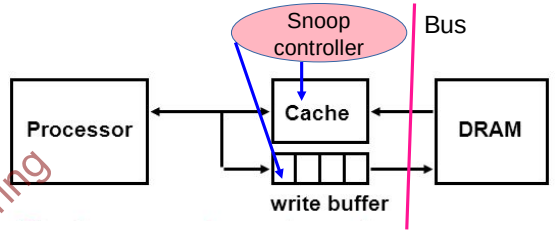
- At block replacement => 2 Bus transactions
  - Bring new block + write-back old block
- Want to reduce processor wait time on a cache miss
- Therefore service the miss first and then do write-back asynchronously
- Need additional storage : write-back buffer

Write buffer



(iii) Dealing with write-backs

- Bus transactions for this block can come. So snooper has to check even the write-back buffer
- Cache controller may have to cancel the write-back, if the block is used from the write-back buffer



Thank you