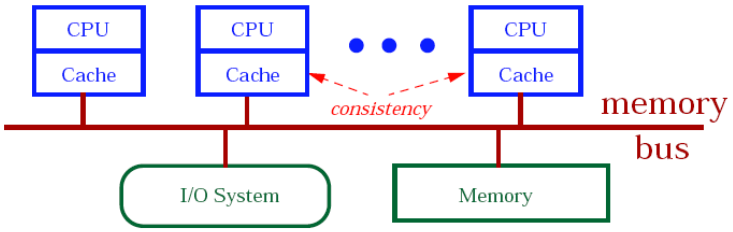
Multiprocessors

Today's topics:

- SMP cache coherence
  - general cache coherence issues
  - snooping protocols
- Improved interaction
  - lots of questions
  - warning – I'm going to wait for answers
    - granted it's an experiment
    - pace will be SLOWer

SMP Review

- Characteristics
  - global physical address space
    - » UMA and hence "symmetric"
  - each processor has it's own cache
    - » for now let's just assume 1 level to simplify things
  - physically shared main memory
    - » easy export of shared memory programming model



Bus Based Coherence

- Cache coherence
  - for shared lines: simple version
    - » all copies of the cached line have the same contents
  - simultaneous update is hard: complex version
    - » for any read: return value of the last write
  - problem: 2 processors write to same value at the same time
    - » how is order determined?
    - » need a single atomic "decider"

Bus Based Coherence

- Cache coherence
  - for shared lines: simple version
    - » all copies of the cached line have the same contents
  - simultaneous update is hard: complex version
    - » for any read: return value of the last write
  - problem: 2 processors write to same value at the same time
    - » how is order determined?
    - » need a single atomic "decider" [Bush'ism ack'd]
- Bus – single thing so it becomes the "decider"
  - limited scalability
    - » even 4 cores is a stretch at today's clock speeds
  - clear broadcast win
    - » all caches see whatever happens on the bus
      - bus order is the write order
      - not good enough then the programmer needs to synchronize

Private vs. Shared Data

- SMP should support both
  - private
    - » normal cache policies and benefits
  - shared: 2 options
    - » NCC-UMA
      - forces all shared data to be via main memory
        - too slow
        - forces programmer to deal with all synchronization
      - requires write- and read-no-allocate instructions
        - otherwise caching could create a problem
        - how?
    - » CC-UMA
      - today's focus
- How to partition shared vs. private?

Private vs. Shared Data

- SMP should support both
  - private
    - » normal cache policies and benefits
  - shared: 2 options
    - » NCC-UMA
      - forces all shared data to be via main memory
        - too slow
        - forces programmer to deal with all synchronization
      - requires write- and read-no-allocate instructions
        - otherwise caching could create a problem
        - how?
    - » CC-UMA
      - today's focus
- How to partition shared vs. private?
  - variable declarations in the code
  - partition by page or segment

Other Sharing Issues

- Consider conventional cache wisdom
  - write-back is good (faster)
    - » problems?
  - large line sizes help exploit spatial locality
    - » problems?
  - valid and dirty tag bits
    - » are they enough?
  - TLB
    - » what changes with page sized partitioning pvt:shared?
  - bus requests
    - » normally always mastered from the cache side
    - » what changes?

Consistency vs. Coherence

- Terminology
  - some confusion in literature
    - » but it's rare so be clear and avoid "mutt" status
  - key is that they are different
- Coherence
  - defines what value is returned by a read
    - » e.g. value of the last write
- Consistency
  - defines when things are coherent
  - bigger issue as systems get bigger
  - *sequential* consistency → value of the last write
    - » as determined by the "decider"
- Both are critical for correctness
  - varies as to whether consistency is exposed to programmer
    - » *sequential* consistency doesn't need to be exposed
      - same as usual sequential programming model

Coherence Implications

- Additional cost
  - caches now need to snoop the bus
    - » watch for writes, tag compare and "update" if they have a copy
      - update options?
- Ordering constraints
  - reordering reads is OK
    - » but not involving writes
      - same as uniprocessor world
  - writes must finish in program order
    - » EVEN if they are independent
      - since there may be a hidden dependency in the other processors
      - also because cache management is by line not variable
    - » this can be relaxed
      - more on this later

2 SMP Protocol Options

- Write-invalidate
  - writer needs exclusive copy
    - » write forces other copies to be invalidated
    - » next read by others is a miss and they get new fresh line
  - 2 writers
    - » one win's bus arbitration and the "decider" has spoken
  - bus broadcast
    - » doesn't need to broadcast write value – only address
- Write-update
  - broadcast write value & address
  - if other copies exist
    - » then appropriate line is updated
- What haven't we considered so far?
  - hint: LOTS

Consider All Cases

- X product
  - (read, write) (miss, hit) (valid copy in cache, memory)
  - (write invalidate, write update)
- Simple with write-through caches
  - memory always has an updated copy
  - new writer gets valid copy
    - » either by cache to cache transfer or from memory
- Harder with write-back caches
  - good idea if cache is mostly holding private data
    - » but memory may not be up to date
      - force invalidate of write back to memory
        - snoop grabs latest copy
      - cache-to-cache copy and no-update of memory
        - If write update and previous owner keeps copy then must clear D bit
        - key: only 1 D-bit can exist max → single "exclusive" owner
- What happens?
  - write miss, read miss

Performance Issues

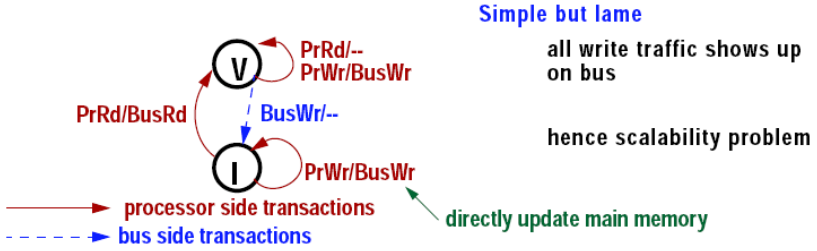
- Too many to exhaustively list
- Key protocol choice issues
  - multiple writes to the same line write invalidate
    - » less bus traffic
      - 1st write → bus invalidate
        - and data transfer on a write miss
      - subsequent writes are kept local
        - as long as there is a write hit
    - » typically Wr-Inv is best choice
      - when line is hammered by one processor at a time
  - write-update
    - » every write generates bus traffic
      - bus scalability is an issue so it easily saturates
    - » still it wins when
      - a certain line is being hammered by multiple processors
        - and when there is 1 writer and the rest are consumers
- Programs share variables not cache lines
  - issues?

Snooping Cache Complexities

- Cache now has 2 masters
  - processor side – same as before
    - » each line still has state I, D tags
      - but add private/shared
    - » tag match the same
      - action can vary
        - write-miss & shared?
        - write-back, clean, and shared?
        - write-back, dirty, and shared?
  - bus side
    - » sees write transactions
      - write-back & dirty?
      - clean?
- End result
  - cache controller FSM now gets more complicated
    - » will vary with
      - cache policy, organization, and share protocol

Simplest Possible Protocol

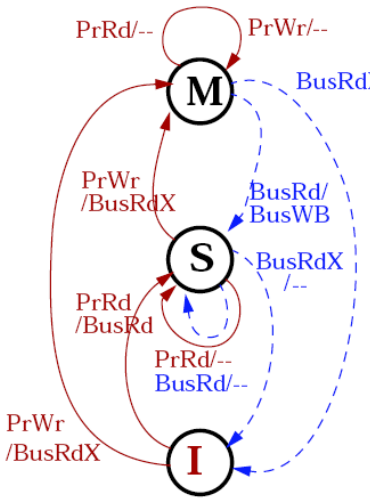
- Write-through, Write-allocate, Write-invalidate
  - Processor transactions: PrRd, PrWr
  - Bus transactions: BusRd, BusWr
  - Line state: I/V
    - » no D bit since write through



MSI Protocol

- Write-invalidate protocal w/ Write-back cache
- Line states: Modified, Shared, Invalid
- Proc side events: PrRd, PrWr
- Bus transactions
  - BusRd – asks for copy of line w/ no intent to modify
    - » e.g. PrRd miss
    - » line supplied by either main memory or another cache
  - BusRdX – asks for exclusive copy of line
    - » PrWr miss or PrWr hit to clean line (not Modified)
    - » note new type of bus transaction
  - BusWB
    - » imposed by write back policy choice
    - » note write data is an entire line

MSI State Diagram



Note:

1: Bus WB supplies data to cache as well as to memory

2. writing to shared block is a problem

BusRdX can be used in 2 situations

Normal

BusUpgr - bus upgrade is one common optimization - in this case no bus occupancy and bandwidth is lost since a data return is not needed for this transaction

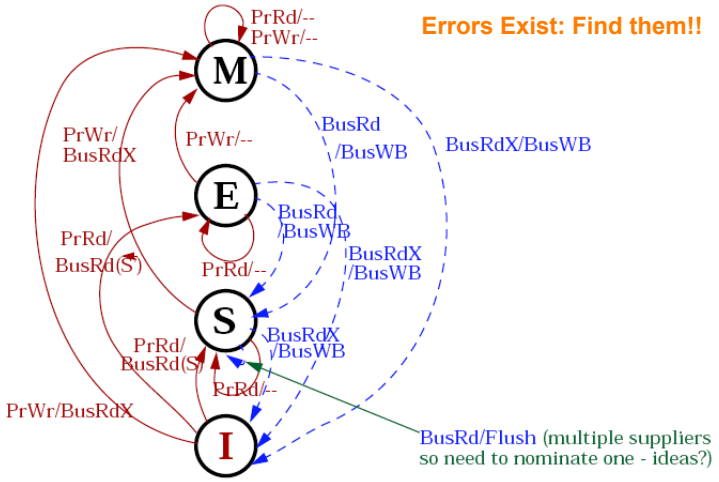
MSI Analysis

- Seq. Consistency
  - write completion
    - » BusRdX & data return complete
  - bus atomicity "decider point" makes this easy
    - » other cache snoopers can see when their pending write is issued when the controller wins arbitration
- Other options
  - BusRd in M: go to I rather than S
    - » migratory protocol – line always just migrates to writer
      - Synapse machine choice
    - » tradeoff
      - another owner likely to write soon then I is better
      - old owner likely to read soon then S is better
    - » hybrid is possible with extra protocol bit
      - choice of Sequent Symmetry and MIT Alewife machines
      - flexibility potential for increased cost & performance
        - – question is how much of each?

MESI Protocol

- Add Exclusive state
  - deals w/ PrRd followed by PrWr problem
  - meanings change a bit
    - » E = exclusive clean – memory is consistent
    - » M = exclusive dirty – memory is inconsistent
    - » S = 2 or more sharers, no writers, memory consistent
    - » I = same as always
- New S semantics adds an additional problem
  - a shared signal must be added to the bus
    - » single wired-OR wire is sufficient
      - note scaling problem – doesn't work well at today's frequencies
    - » BusRd(S) – shared signal asserted
    - » BusRd(S') – shared signal not asserted
    - » BusRd – means don't care about shared signal
    - » FLUSH – optional for cache to cache copy?

MESI State Machine



Errors Exist: Find them!!

MESI Analysis

- **Flush issues**
  - **don't want redundant suppliers when a new sharer comes on line**
    - » **last exclusive owner knows who they are**
    - » **so that one does the flush**
    - » **if no sharing then supplied from memory**
  - **complicates bus**
  - **Stanford Dash & SGI Origin series choice**
- **What haven't we worried about yet?**

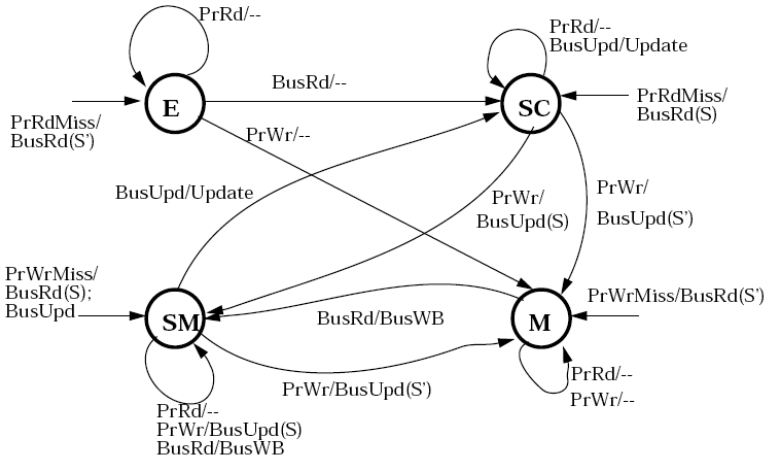
MESI Analysis

- Flush issues
  - don't want redundant suppliers when a new sharer comes on line
    - » last exclusive owner knows who they are
    - » so that one does the flush
    - » if no sharing then supplied from memory
  - complicates bus
  - Stanford Dash & SGI Origin series choice
- What haven't we worried about yet?
  - what happens when a line gets victimized?
    - » exercise to figure out the new state machine

Dragon Protocol

- Write-back and Write-update*
  - Xerox PARC Dragon
    - » subsequently modified somewhat for Sun's SparcServer
  - states
    - » E – exclusive clean
    - » SC – shared clean
    - » SM – shared modified – this one used to update memory
    - » M – exclusive dirty
    - » no explicit I state – there implicitly
  - new bus transactions
    - » BusUpd – update request with same S and S' variants

Dragon FSM



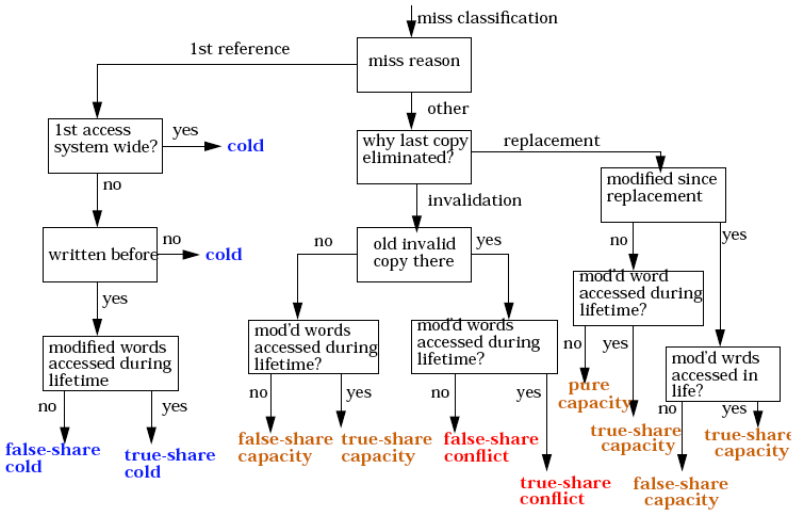
Key Points

- **Status tags**
  - **need to encode the local line status**
    - » **protocol dependent**
- **2 ported cache controller**
  - **priority becomes the bus**
    - » **since it's the atomicity point**
  - **possibly stalls process requests**
- **New miss source**
  - **the 4th C: Coherence**
    - » **true shared miss: reads and writes to same target**
    - » **false shared miss: reads and writes to different target but same line**
- **Increased bus pressure**
  - **due to coherence traffic**
    - » **increased power**
  - **already a scaling problem**

Classifying Misses

- **For a particular reference stream**
  - **define the lifetime for a block in the cache**
  - **do per word accounting**
    - » **e.g. remote reference from processor x causes eviction**
- **Ideas for how to do this?**

Miss Classification



Getting More Real

- 2 level cache hierarchy is likely
  - Harvard L1$
  - Unified L2
- L2 is the one that sits on the bus now
  - L2 is coherent but what about L1's
    - » L1 write and read misses don't cause much problem
      - percolate through to L2 and then the rest is similar
    - » L1 read hit – no problem
    - » L1 write hit
      - write has to percolate all the way to the bus
  - L2 line eviction
    - » due to invalidate
    - » L2 needs to pass eviction up and evict L1's entry
- More synchronization through the cache hierarchy
  - will slow things down
    - » question is how much

Concluding Remarks

- How well does it work
  - see Chap. 4.3 data
- Is SMP dead because buses are dead?
  - small way SMP may make sense
    - » on multi-core socket
    - » or in small clusters where socket is multi-cluster
  - short buses aren't so bad
    - » easy enough to extend life with point to point interconnect
- Next we move onto DSM variant of CC-Numa
  - protocol ideas are still valid
    - » hence the time spent to understand these protocols is well spent
      - note exam question is highly likely
  - main difference with DSM
    - » lines have both
      - local state: similar to today's discussion
      - global state: more on that next lecture