Computer Organization

Part 25 : MESI Protocol

UNIT – IV

Tushar B. Kute,
Department of Information Technology,
Sandip Institute of Technology & Research Centre, Nashik.
http://tusharkute.com
Problem of Memory Coherence

- Assume just single level caches and main memory
- Processor writes to location in its cache
- Other caches may hold shared copies - these will be out of date
- Updating main memory alone is not enough
Processor 1 reads X: obtains 24 from memory and caches it
Processor 2 reads X: obtains 24 from memory and caches it
Processor 1 writes 32 to X: its locally cached copy is updated
Processor 3 reads X: what value should it get?
   Memory and processor 2 think it is 24
   Processor 1 thinks it is 32

Notice that having write-through caches is not good enough
Bus Snooping

• Scheme where every CPU knows who has a copy of its cached data is far too complex.
• So each CPU (cache system) ‘snoops’ (i.e. watches continually) for write activity concerned with data addresses which it has cached.
• This assumes a bus structure which is ‘global’, i.e. all communication can be seen by all.
• More scalable solution: ‘directory based’ coherence schemes
Snooping Protocols

• Write Invalidate
  – CPU wanting to write to an address, grabs a bus cycle and sends a ‘write invalidate’ message
  – All snooping caches invalidate their copy of appropriate cache line
  – CPU writes to its cached copy (assume for now that it also writes through to memory)
  – Any shared read in other CPUs will now miss in cache and re-fetch new data.
Snooping Protocols

• Write Update
  – CPU wanting to write grabs bus cycle and broadcasts new data as it updates its own copy
  – All snooping caches update their copy

• Note that in both schemes, problem of simultaneous writes is taken care of by bus arbitration - only one CPU can use the bus at any one time.
Update or Invalidate?

• Update looks the simplest, most obvious and fastest, but:
  – Multiple writes to same word (no intervening read) need only one invalidate message but would require an update for each
  – Writes to same block in (usual) multi-word cache block require only one invalidate but would require multiple updates.
Update or Invalidate?

• Due to both spatial and temporal locality, previous cases occur often.
• Bus bandwidth is a precious commodity in shared memory multi-processors
• Experience has shown that invalidate protocols use significantly less bandwidth.
• Will consider implementation details only of invalidate.
Implementation Issues

• In both schemes, knowing if a cached value is not shared (copy in another cache) can avoid sending any messages.

• Invalidate description assumed that a cache value update was written through to memory. If we used a ‘copy back’ scheme other processors could re-fetch old value on a cache miss.

• We need a protocol to handle all this.
MESI Protocol (1)

• A practical multiprocessor invalidate protocol which attempts to minimize bus usage.
• Allows usage of a ‘write back’ scheme - i.e. main memory not updated until ‘dirty’ cache line is displaced
• Extension of usual cache tags, i.e. invalid tag and ‘dirty’ tag in normal write back cache.
MESI Protocol (2)

Any cache line can be in one of 4 states (2 bits)

• **Modified** - cache line has been modified, is different from main memory - is the only cached copy. (multiprocessor ‘dirty’)

• **Exclusive** - cache line is the same as main memory and is the only cached copy

• **Shared** - Same as main memory but copies may exist in other caches.

• **Invalid** - Line data is not valid (as in simple cache)
MESI Protocol (3)

• Cache line changes state as a function of memory access events.

• Event may be either
  – Due to local processor activity (i.e. cache access)
  – Due to bus activity - as a result of snooping

• Cache line has its own state affected only if address matches
MESI Protocol (4)

• Operation can be described informally by looking at action in local processor
  – Read Hit
  – Read Miss
  – Write Hit
  – Write Miss

• More formally by state transition diagram
MESI Local Read Hit

- Line must be in one of MES
- This must be correct local value (if M it must have been modified locally)
- Simply return value
- No state change
MESI Local Read Miss (1)

• No other copy in caches
  – Processor makes bus request to memory
  – Value read to local cache, marked E

• One cache has E copy
  – Processor makes bus request to memory
  – Snooping cache puts copy value on the bus
  – Memory access is abandoned
  – Local processor caches value
  – Both lines set to S
MESI Local Read Miss (2)

- Several caches have S copy
  - Processor makes bus request to memory
  - One cache puts copy value on the bus (arbitrated)
  - Memory access is abandoned
  - Local processor caches value
  - Local copy set to S
  - Other copies remain S
MESI Local Read Miss (3)

- One cache has M copy
  - Processor makes bus request to memory
  - Snooping cache puts copy value on the bus
  - Memory access is abandoned
  - Local processor caches value
  - Local copy tagged S
  - Source (M) value copied back to memory
  - Source value M -> S
MESI Local Write Hit (1)

Line must be one of MES

- **M**
  - line is exclusive and already ‘dirty’
  - Update local cache value
  - no state change

- **E**
  - Update local cache value
  - State E -> M
MESI Local Write Hit (2)

- S
  - Processor broadcasts an invalidate on bus
  - Snooping processors with S copy change S->I
  - Local cache value is updated
  - Local state change S->M
MESI Local Write Miss (1)

Detailed action depends on copies in other processors

- No other copies
  - Value read from memory to local cache (?)
  - Value updated
  - Local copy state set to M
MESI Local Write Miss (2)

- Other copies, either one in state E or more in state S
  - Value read from memory to local cache - bus transaction marked RWITM (read with intent to modify)
  - Snooping processors see this and set their copy state to I
  - Local copy updated & state set to M
MESI Local Write Miss (3)

Another copy in state M

- Processor issues bus transaction marked RWITM
- Snooping processor sees this
  - Blocks RWITM request
  - Takes control of bus
  - Writes back its copy to memory
  - Sets its copy state to I
MESI Local Write Miss (4)

Another copy in state M (continued)

- Original local processor re-issues RWITM request
- Is now simple no-copy case
  - Value read from memory to local cache
  - Local copy value updated
  - Local copy state set to M
Putting it all together

• All of this information can be described compactly using a state transition diagram

• Diagram shows what happens to a cache line in a processor as a result of
  – memory accesses made by that processor (read hit/miss, write hit/miss)
  – memory accesses made by other processors that result in bus transactions observed by this snoopy cache (Mem read, RWITM, Invalidate)
MESI – locally initiated accesses

- Invalid
- Shared
- Modified
- Exclusive

- Read Miss (sh)
- Read Miss (ex)
- Invalidate
- Mem Read
- Write Miss
- Read Hit
- Write Hit

= bus transaction
Reference

• Computer Architecture and Organization
  – By A. P. Godse

• Computer Organization
  – By Hamacher and Zaky