Lecture 9 Outline
- MESI protocol
- Dragon update-based protocol
- Impact of protocol optimizations

MESI (4-state) Invalidation Protocol
- Problem with MSI protocol
  - Rd, Wr sequence incurs 2 transactions
    - even when no one is sharing (e.g., serial program?)
    - BusRd (I → S) followed by BusRdX or BusUpgr (S → M)
  - In general, penalizing serial programs is unacceptable
- Add exclusive state:
  - Invalid
  - Modified (dirty)
  - Shared (two or more caches may have copies)
  - Exclusive: (only this cache has clean copy, same value as in memory)
- How to decide I → E or I → S?
  - Need to check whether someone else has copy
    - "Shared" signal on bus: wired-or line asserted in response to BusRd

MESI: Processor-Initiated Transactions

MESI: Bus-Initiated Transactions

MESI State Transition Diagram
Flush vs. Flush1 (Flush’ in textbook)

- Flush: mandatory
- Flush' (Flush1): happens only when
  - Cache-to-cache sharing is used, and,
  - Only one cache flushes data

MESI Visualization

One less bus request due to Exclusive state, esp. for serial programs
MESI Visualization

P1

X=1

S

Snooper

Flush

Mem Ctrl

X=2

P2

X=2

S

Snooper

P3

X=2

S

Snooper

Note: BusUpgr instead of BusRdX

Lecture 9

ECE/CSC 506 - Summer 2006 - E. F. Gehringer, based on slides by Yan Solihin

 MESI Visualization

P1

X=3

S

Snooper

Mem Ctrl

X=3

P2

X=3

S

Mem Ctrl

X=3

P3

X=3

S

rd & X

BusRd

P1/P3

Cache

BusRd/Flush

S-SR

P3 cache

BusRd/Flush

S-SR

P1 cache

BusRd/Flush

S-SR

Mem

BusRdX

M

I

Proc

Action

State P1

State P2

State P3

Bus Action

Data From

R1

E

-

-

BusRd

Mem

W1

M

-

-

Own cache

R3

S

-

S

BusRd/Flush

P1 cache

W3

I

-

M

BusRdX

Mem

R1

S

-

S

BusRd/Flush

P3 cache

R3

S

-

S

Own cache

R2

S

S

S

BusRd/Flush

P1/P3

Cache

* Data from memory if no cache2cache transfer, BusRd/-

Lecture 9

ECE/CSC 506 - Summer 2006 - E. F. Gehringer, based on slides by Yan Solihin


**MESI Example (Cache-to-Cache Transfer + BusUpgr)**

<table>
<thead>
<tr>
<th>Proc Action</th>
<th>State P1</th>
<th>State P2</th>
<th>State P3</th>
<th>Bus Action</th>
<th>Data From</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>E</td>
<td>-</td>
<td>-</td>
<td>BusRd</td>
<td>Mem</td>
</tr>
<tr>
<td>W1</td>
<td>S</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Own cache</td>
</tr>
<tr>
<td>R3</td>
<td>S</td>
<td>-</td>
<td>M</td>
<td>BusRd/Flush</td>
<td>P1 cache</td>
</tr>
<tr>
<td>W3</td>
<td>I</td>
<td>-</td>
<td>M</td>
<td>BusUpgr</td>
<td>Own cache</td>
</tr>
<tr>
<td>R3</td>
<td>S</td>
<td>-</td>
<td>S</td>
<td>BusRd/Flush</td>
<td>P3 cache</td>
</tr>
<tr>
<td>R3</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>BusRd/Flush</td>
<td>P1/P3 cache</td>
</tr>
</tbody>
</table>

* Data from memory if no cache2cache transfer, BusRd/-

**Lower-Level Protocol Choices**

- Who supplies data on miss when not in M state: memory or cache?
- Original, Illinois MESI: cache
  - Assume cache faster than memory (Cache-to-cache transfer)
  - Not necessarily true
- Adds complexity
  - How does memory know it should supply data? (must wait for caches)
  - Selection algorithm if multiple caches have valid data
- Valuable for distributed memory
  - May be cheaper to obtain from nearby cache than distant memory
  - Especially when constructed out of SMP nodes (Stanford DASH)

**Lecture 9 Outline**

- MESI protocol
- Dragon update-based protocol
- Impact of protocol optimizations

**Dragon Writeback Update Protocol**

- Four states
  - Exclusive-clean (E): I and memory have it
  - Shared clean (Sc): I, others, and maybe memory, but I'm not owner
  - Shared modified (Sm): I and others but not memory, and I'm the owner
    - Sm and Sc can coexist in different caches, with at most one Sm
  - Modified or dirty (M): I and, no one else
- On replacement: Sc can silently drop, Sm has to flush
- No invalid state
  - In cache, cannot be invalid
  - If not present in cache, can view as being in not-present or invalid state
- New processor events: PrWrMiss, PrWMMiss
- Introduced to specify actions when block not present in cache
- New bus transaction: BusUpd
  - Broadcasts single word written on bus; updates other relevant caches

**Dragon State Transition Diagram**

**Dragon: Processor-Initiated Transactions**

- Processor-Initiated Transactions
- PrRd/-
- PrWr/-
- PrWr/BusUpd(S)
- PrWr/BusUpd(~S)
- PrWrMiss/BusRd(S)
- PrWMMiss/BusRd(S)
- PrWrMiss/BusUpd(S)
- PrWMMiss/BusUpd(S)
- PrWMMiss/BusUpd(~S)
- PrWrMiss/BusUpd(~S)
- PrWMMiss/BusUpd(~S)
- PrWrRd/PrWr/PrWr/-
Dragon: Bus-Initiated Transactions

- BusRd/Flush
- BusUpd/Update
- BusRd/-BusUpd/Update

Dragon Visualization

- P1
- P2
- P3
- Snooper
- Cache
- Main Memory
- Bus
- Mem Ctrl

Dragon Visualization

- X=1
- Mem Ctrl
- X=1
- Mem Ctrl

Dragon Visualization

- X=1
- Mem Ctrl
- X=2
- Mem Ctrl

One less bus request due to Exclusive state, esp. for serial programs

Dragon Visualization

- X=1
- Mem Ctrl
- X=2
- Mem Ctrl

rd &X

wr &X (X=2)

One less bus request due to Exclusive state, esp. for serial programs
This is a miss in the MESI and MSI protocols.

Note: only one with Sm is responsible for cache-to-cache transfer.

Note: BusUpdate instead of BusUpgr (no inval is performed).

P1 replaces X.
Lower-Level Protocol Choices

- Can shared-modified state be eliminated?
  - If update memory as well on BusUpd transactions (DEC Firefly)
  - Dragon protocol doesn’t (assumes DRAM memory slow to update)
- Should replacement of an S block be broadcast?
  - Would allow last copy to go to Exclusive state and not generate updates
  - Replacement bus transaction is not in critical path; later update may be
- Coherence, consistency considerations much like write-through case
  - In general, many subtle race conditions in protocols
  - But first, let’s illustrate quantitative assessment at logical level

Assessing Protocol Tradeoffs

- Methodology:
  - Use simulator; choose parameters per earlier methodology
    (default 1MB, 4-way cache, 64-byte block, 16 processors; 64K cache for some)
  - Focus on frequencies, not end performance for now
    - Transcends architectural details, but not what we’re really after
  - Use idealized memory performance model to avoid changes of reference interweaving across processors with machine parameters
    - Cheap simulation: no need to model contention

Impact of Protocol Optimizations

- MSI vs. MESI where write-back only when the line is in M state

Dragon Visualization

Dragon Example

<table>
<thead>
<tr>
<th>Proc Action</th>
<th>State P1</th>
<th>State P2</th>
<th>State P3</th>
<th>Bus Action</th>
<th>Data From</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>E</td>
<td>-</td>
<td>-</td>
<td>BusRd</td>
<td>Mem</td>
</tr>
<tr>
<td>W1</td>
<td>M</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Own cache</td>
</tr>
<tr>
<td>R3</td>
<td>Sm</td>
<td>-</td>
<td>Sc</td>
<td>BusRd/Flush</td>
<td>Own cache</td>
</tr>
<tr>
<td>W3</td>
<td>Sc</td>
<td>-</td>
<td>Sm</td>
<td>BusUpd/Updt</td>
<td>Own cache</td>
</tr>
<tr>
<td>R1</td>
<td>Sc</td>
<td>-</td>
<td>Sm</td>
<td>-</td>
<td>Own cache</td>
</tr>
<tr>
<td>R3</td>
<td>Sc</td>
<td>-</td>
<td>Sm</td>
<td>-</td>
<td>Own cache</td>
</tr>
<tr>
<td>R2</td>
<td>Sc</td>
<td>Sc</td>
<td>Sm</td>
<td>BusRd/Flush</td>
<td>P3 cache</td>
</tr>
</tbody>
</table>
Impact of Cache-Block Size

- Multiprocessors add new kind of miss to cold, capacity, conflict
  - Coherence misses: Due to invalidations
    - True sharing: Write to same word
    - False sharing: Write to different words
  - Reducing misses architecturally in invalidation protocol
    - Capacity: enlarge cache; increase block size (if spatial locality)
    - Conflict: increase associativity
    - Cold and coherence: only block size
  - Increasing block size has advantages and disadvantages
    - Can reduce misses if spatial locality is good
    - Can hurt too
      - Increase misses due to false sharing if spatial locality not good
      - Increase misses due to conflicts in fixed-size cache
      - Increase traffic due to fetching unnecessary data and due to false sharing
      - Can increase miss penalty and perhaps hit cost

Impact of Block Size on Miss Rate

- For default problem size: vary block/line size from 8-256 Bytes
  - Decreases with larger lines: cold, capacity (due to spatial locality), true sharing (due to spatial locality)
  - Increases with larger lines: false sharing
  - Working set doesn’t fit: impact of capacity misses large: (Ocean, Radix)

Impact of Block Size on Traffic

Traffic (bytes/instruction) affects performance indirectly through contention

- Results different than for miss rate: traffic almost always increases
- When working sets fit, overall traffic still small, except for Radix
  - Fixed overhead is significant component
  - So total traffic often minimized at 16-32 byte block, not smaller
- Working set doesn’t fit: even 128-byte good for Ocean due to capacity
  - Address bus traffic behaves in opposite way as the data bus traffic