CIS 429/529

Cache Coherence Protocols

Snoopy Protocol

Prof. Michel A. Kinsy
- Use snoopy mechanism to keep all processors’ view of memory coherent
• The MSI protocol

Each cache line has a tag

<table>
<thead>
<tr>
<th>State Bits</th>
<th>Address Tag</th>
</tr>
</thead>
</table>

M: Modified  
S: Shared  
I: Invalid

Read miss  
Read by any processor

Other processor reads  
P₁ writes back

P₁ intends to write

Other processor intends to write  
P₁ writes back

Cache state in processor P₁

P₁ reads or writes

Write miss
MESI: An Enhanced MSI protocol

Each cache line has a tag

- **M**: Modified Exclusive
- **E**: Exclusive, unmodified
- **S**: Shared
- **I**: Invalid

<table>
<thead>
<tr>
<th>State bits</th>
<th>Address tag</th>
</tr>
</thead>
</table>

Cache state in processor $P_1$

- **Read miss, not shared**
- **Write miss**
- **Other processor intent to write**

Rules:

- **P₁ write or read**
- **Other processor reads $P₁$ writes back**
- **Read miss, shared**
- **Read by any processor**
- **P₁ intent to write**
2 Processor Example

Block b

P₁ write or read

P₂ reads,
P₁ writes back

Read miss

P₁ write

P₂ intent to write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ intent to write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
P₁ writes back

P₁ intent to write

P₂ write

P₂ reads,
Cache Coherence State Encoding

Valid and dirty bits can be used to encode $S$, $I$, and $(E, M)$ states:
- $V=0$, $D=x \rightarrow$ Invalid
- $V=1$, $D=0 \rightarrow$ Shared (not dirty)
- $V=1$, $D=1 \rightarrow$ Exclusive (dirty)
2-Level On-chip Caches

- **Inclusion property**: entries in L1 must be in L2
  - invalidation in L2 $\rightarrow$ invalidation in L1

- **Does snooping on L2 affect CPU-L1 bandwidth?**
  - yes -- to check if a dirty copy is stored in L1

- **How can this be avoided?**
  - Write-through L1 cache
Intervention

- When a read-miss for A occurs in cache-2, a read request for A is placed on the bus
  - Cache-1 needs to supply & change its state to shared
  - The memory may respond to the request also!
Intervention

- Does memory know it has stale data?
  - Cache-1 needs to intervene through memory controller to supply correct data to cache-2
False Sharing

A cache block contains more than one word

Cache-coherence is done at the block-level and not word-level

Suppose \( M_1 \) writes \( \text{word}_i \) and \( M_2 \) writes \( \text{word}_k \) and both words have the same block address.

What can happen?

- Block may be invalidated many times unnecessarily
• Performance Issues

- Cache-coherence protocols will cause mutex to ping-pong between P1’s and P2’s caches.
- Ping-ponging can be reduced by first reading the mutex location (non-atomically) and executing a swap only if it is found to be zero.
Performance Related to Bus Occupancy

• In general, a read-modify-write instruction requires two memory (bus) operations without intervening memory operations by other processors

• In a multiprocessor setting, bus needs to be locked for the entire duration of the atomic read and write operation
  ▶ expensive for simple buses
  ▶ very expensive for split-transaction buses

• modern processors use
  ▶ load-reserve and store-conditional
Load-reserve & Store-conditional

• Special register(s) to hold reservation flag and address, and the outcome of store-conditional

Load-reserve R, (a):
<flag, addr> ← <1, a>;
R ← M[a];

Store-conditional (a), R:
if <flag, addr> == <1, a> then
cancel other procs’ reservation on a;
M[a] ← <R>;
status ← succeed;
else status ← fail;

• If the snooper sees a store transaction to the address in the reserve register, the reserve bit is set to 0

▷ Several processors may reserve ‘a’ simultaneously
▷ These instructions are like ordinary loads and stores with respect to the bus traffic
Performance

• Load-reserve & Store-conditional
  
  ‣ The total number of memory (bus) transactions is not necessarily reduced, but splitting an atomic instruction into load-reserve & store-conditional:

  • increases bus utilization (and reduces processor stall time), especially in split-transaction buses

  • reduces cache ping-pong effect because processors trying to acquire a semaphore do not have to perform stores each time
Maintaining Cache Coherence

• Hardware support is required such that
  ‣ only one processor at a time has write permission for a location
  ‣ no processor can load a stale copy of the location after a write

• write request:
  ‣ The address is invalidated in all other caches before the write is performed

• read request:
  ‣ If a dirty copy is found in some cache, a write-back is performed before the memory is read
Software Cache Coherence

- Exclude hardware support for cache coherence, e.g., Cray T3D
- Systems with caches mark shared data as uncachable
- Software can explicitly cache value of stored data
- Disadvantages?
  - Compiler mechanisms for CC very limited
  - Need to be conservative: every block that might be shared is treated as shared
  - Doing things at the cache block level more efficient