

CS533: Scalable Cache Coherence

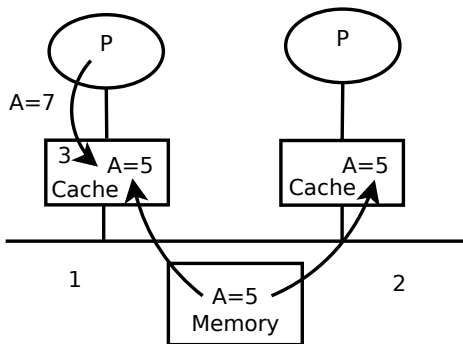
Josep Torrellas

University of Illinois in Urbana-Champaign

- * Review: Chapter on Cache Coherence from any of the two textbooks
- * A. Gupta, W.-D. Weber, and T. Mowry.
Reducing memory and traffic requirements for scalable directory-based cache coherence schemes.
In International Conference on Parallel Processing, 1990.
- * D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. Hennessy.
The directory-based cache coherence protocol for the dash multiprocessor.
In International Symposium on Computer Architecture, 1990.
- * J. Torrellas, M. S. Lam, and J. L. Hennessy.
False sharing and spatial locality in multiprocessor caches.
In Transactions on Computers, June 1994.

The Cache Coherence Problem

- Caches are critical to modern high-speed processors
- Multiple copies of a block can easily get inconsistent
 - processor writes, I/O writes, ...



Cache Coherence Solutions

- Software based vs hardware based
- Software-based:
 - Compiler based or with run-time system support
 - With or without hardware assist
 - Tough problem because perfect information is needed in the presence of memory aliasing and explicit parallelism
- Focus on hardware based solutions as they are more common

The schemes can be classified based on:

- Snoopy schemes vs Directory schemes
- Write-through vs Write-back (ownership-based) protocols
- Update vs Invalidation protocols
- Dirty-sharing vs No-dirty-sharing protocols

Snoopy Cache Coherence Schemes

- A distributed cache coherence scheme based on the notion of a snoop
 - watches all activity on a global bus
 - or is informed about such activity by some global broadcast mechanism
-

Write-Back/Ownership Schemes

- When a single cache has ownership of a block, processor writes do not result in bus writes thus conserving bandwidth
- Most bus-based multiprocessors nowadays use such schemes
- Many variants of ownership-based protocols exist

Write Through Schemes

All processor writes result in:

- update of local cache and a global bus write that:
 - 1 updates main memory
 - 2 invalidates/updates all other caches with that item

Not used because too much traffic

Invalidation vs Update Strategies

- ❶ Invalidation: On a write, all other caches with a copy are invalidated
- ❷ Update: On a write, all other caches with a copy are updated

Invalidation is bad when:

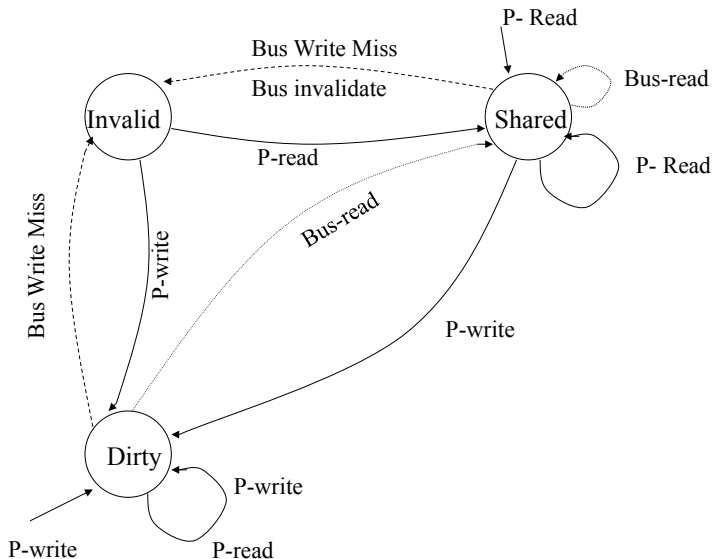
- producer and (one or more) consumers of data

Update is bad when:

- multiple writes by one PE before data is read by another PE
- Junk data accumulates in large caches (e.g. process migration)

Invalidation schemes are used

Base Invalidation Protocol (MSI)



Illinois Scheme (MESI)

- States: I, VE (valid-exclusive), VS (valid-shared), D (dirty)
- The cache knows if it has a valid-exclusive (VE) copy
- In VE state no invalidation traffic on write-hits

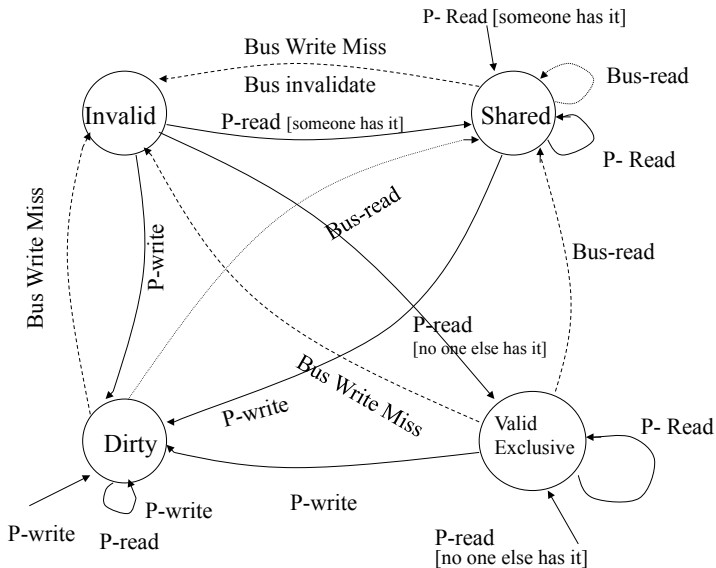
Advantage:

- Closely approximates traffic on a uniprocessor for sequential programs

Disadvantage:

- More complex

Illinois Scheme (MESI)



DEC Firefly Scheme

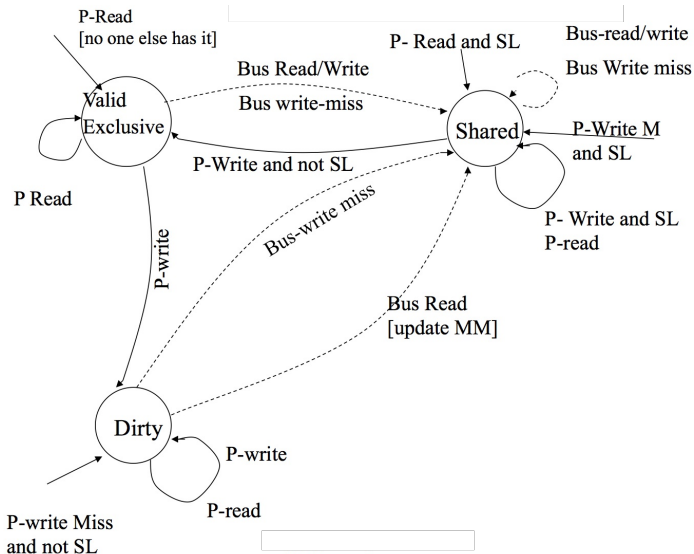
Classification: Write-back, update, no-dirty-sharing

States:	State	Explanation
	VE (valid exclusive)	only copy and clean
	VS (valid shared)	shared-clean copy
	D (Dirty)	dirty exclusive (only copy)

- VS write hits result in updates to memory and other caches and entry remains in this state
- Used special “shared line” on bus to detect sharing status of cache line
- Supports producer-consumer model well

What about sequential processes migrating between CPU's?

DEC Firefly Scheme



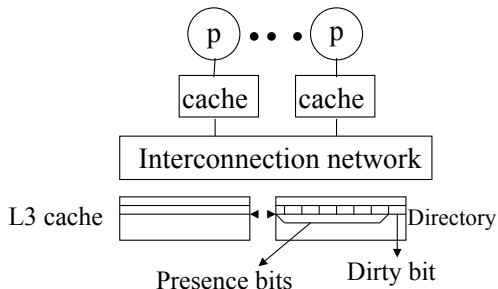
Directory Based Cache Coherence

Key idea: keep track in a global directory (in main memory) of which processors are caching a location and the state

- Motivation: Snoopy schemes do not scale because they rely on broadcast
- Directory based schemes allow scaling
 - They avoid broadcasts by keeping track of all PEs caching a memory block, and then using point-to-point messages to maintain coherence
 - They allow the flexibility to use any scalable point-to-point network

Basic Scheme (Censier and Feautrier)

- Assume K processors
- With each cache-block in memory: K presence bits and 1 dirty bit



Read Miss

Read from LLC by PE_i

If dirty bit is OFF then:

- 1 read from LLC;
- 2 turn $p[i]$ ON;

if dirty bit is ON then:

- 1 recall line from dirty PE (cache state to shared);
- 2 update LLC;
- 3 turn dirty-bit OFF;
- 4 turn $p[i]$ ON;
- 5 supply called data to PE_i

Write Miss

if dirty-bit is OFF then:

- 1 send invalidatons to all PE's caching that block and clear their $P[k]$ bits;
- 2 turn dity bit ON;
- 3 turn $P[i]$ ON
- 4 supply data to PE_i ;

if dirty-bit is ON then:

- 1 recall the data from owner PE which invalidates itself
- 2 update LLC
- 3 clear bit of previous owner
- 4 turn bit $PE[i]$ on (dirty bit ON all the time)
- 5 forward data to PE_i ;

Write Hit to Non-Owned Data

Write-hit to data valid(not owned) in cache:

- 1 access LLC-directory;
- 2 send invalidations to all PE's caching block;
- 3 clear their $P[k]$ bits;
- 4 turn dirty bit ON;
- 5 turn $PE[i]$ ON;
- 6 supply data to PE_i

Scaling of LLC and directory bandwidth

- Cannot have LLC memory or directory memory centralized
- Need a distributed cache coherence protocol

As shown, directory memory requirements do not scale well

- Reason is that the number of presence bits needed grows as the number of PEs increases.
- Also: the larger the LLC is, the larger the directory

Directory Organizations

- Memory-based schemes (DASH) vs Cache-based schemes (SCI)
- Cache-based schemes (or linked-list based)
 - Singly linked
 - Doubly-linked (SCI)
- Memory-based schemes (or pointer-based)
 - Full map (Dir-N) vs Partial-map schemes (Dir-i-B, Dir-i-CV-r, ...)
 - Dense (DASH) vs Sparse directory schemes

Pointer-Based Coherence Schemes

- The Full Bit Vector Scheme
- Limited Pointer Schemes
- Sparse Directories (Caching)
- LimitLess (Software Assistance)

The Full Bit Vector Scheme

- One bit of directory memory per LLC block per PE
- Memory requirements are $P \times (P \times M/B)$, where P is the number of PEs, M is LLC per PE, and B is cache block size (not counting the dirty bit)
- Invalidation traffic is best
- One way to reduce the overhead is to increase B
 - Can result in false sharing and increased coherence traffic
- Overhead may be ok for modest-scale mps
 -

Limited Pointer Schemes

- Since data is expected to be in only a few caches at any one time, a limited number of pointers per directory entry should suffice
- Overflow strategy: what to do when the number of sharers exceeds the number of points?
- Many different schemes based on different overflow strategies

Some Examples

Dir-i-B

- Beyond i-pointers, set the inval-broadcast bit ON
- Storage needed is: $i \times \log(P) \times P \times M/B$ (in addition to inval-broadcast bit)
- Expected to do well since widely shared data is not written often

Dir-i-NB

- When sharers exceed i , invalidate one of the existing sharers
- Significant degradation expected for widely-shared, mostly-read data

Dir-i-CV-r

- When sharers exceed i , use bits allocated to i pointers as a coarse resolution vector (each bit points to multiple PEs)
- Always results in less coherence traffic than Dir-i-B
- Example: use Dir-3-CV-4 for 64 processors

Performance of Directories

- Figure 10 in Gupta *et al.* paper
- Figure 7 in Gupta *et al.* paper

LimitLess Directories

- Limit number of pointers
- On overflow:
 - Memory module interrupts the local processor
 - Processor emulates the full-map directory for block

LimitLess Directories Requirements

- Rapid trap handler: trap code executes within 5-10 cycles from trap initiation
- Software has complete access to coherence controller
- Interface to the network that allows the processor to launch and intercept coherence protocol packets