CSE 490/590 Computer Architecture

Directory-Based Caches II

Steve Ko
Computer Sciences and Engineering
University at Buffalo

Last time…

- True sharing vs. false sharing
- Miss vs. hit in multiprocessors

A Cache Coherent System Must:

- Provide set of states, state transition diagram, and actions
- Manage coherence protocol
  - (0) Determine when to invoke coherence protocol
  - (a) Find info about state of address in other caches to determine action
    - » whether need to communicate with other cached copies
  - (b) Locate the other copies
  - (c) Communicate with those copies (invalidate/update)
- (0) is done the same way on all systems
  - state of the line is maintained in the cache
  - protocol is invoked if an "access fault" occurs on the line
- Different approaches distinguished by (a) to (c)

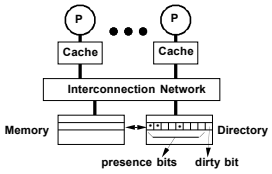
Bus-based Coherence

- All of (a), (b), (c) done through broadcast on bus
  - faulting processor sends out a "search"
  - others respond to the search probe and take necessary action
- Could do it in scalable network too
  - broadcast to all processors, and let them respond
- Conceptually simple, but broadcast doesn't scale with number of processors, P
  - on bus, bus bandwidth doesn't scale
  - on scalable network, every fault leads to at least P network transactions
- Scalable coherence:
  - can have same cache states and state transition diagram
  - different mechanisms to manage protocol

Scalable Approach: Directories

- Every memory block has associated directory information
  - keeps track of copies of cached blocks and their states
  - on a miss, find directory entry, look it up, and communicate only with the nodes that have copies if necessary
  - in scalable networks, communication with directory and copies is through network transactions
- Many alternatives for organizing directory information

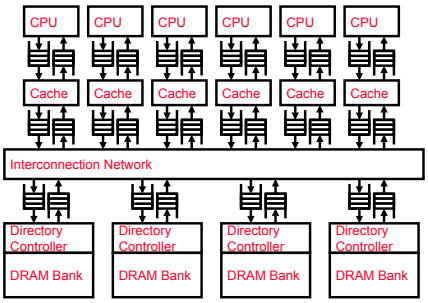
Basic Operation of Directory



- k processors.
- With each cache-block in memory: k presence-bits, 1 dirty-bit
- With each cache-block in cache: 1 valid bit, and 1 dirty (owner) bit

- Read from main memory by processor i:
  - If dirty-bit OFF then { read from main memory; turn p[i] ON; }
  - if dirty-bit ON then { recall line from dirty proc (downgrade cache state to shared); update memory; turn dirty-bit OFF; turn p[i] ON; supply recalled data to i;}
- Write to main memory by processor i:
  - If dirty-bit OFF then {send invalidations to all caches that have the block; turn dirty-bit ON; supply data to i; turn p[i] ON; ... }

Directory Cache Protocol



- Assumptions: Reliable network, FIFO message delivery between any given source-destination pair

Cache States

For each cache line, there are 4 possible states:

- C-invalid (= Nothing): The accessed data is not resident in the cache.
- C-shared (= Sh): The accessed data is resident in the cache, and possibly also cached at other sites. The data in memory is valid.
- C-modified (= Ex): The accessed data is exclusively resident in this cache, and has been modified. Memory does not have the most up-to-date data.
- C-transient (= Pending): The accessed data is in a *transient* state (for example, the site has just issued a protocol request, but has not received the corresponding protocol reply).

Home directory states

- For each memory block, there are 4 possible states:
  - R(dir): The memory block is shared by the sites specified in dir (dir is a set of sites). The data in memory is valid in this state. If dir is empty (i.e., dir = ε), the memory block is not cached by any site.
  - W(id): The memory block is exclusively cached at site id, and has been modified at that site. Memory does not have the most up-to-date data.
  - TR(dir): The memory block is in a transient state waiting for the acknowledgements to the invalidation requests that the home site has issued.
  - TW(id): The memory block is in a transient state waiting for a block exclusively cached at site id (i.e., in C-modified state) to make the memory block at the home site up-to-date.

CSE 490/590 Administrivia

- Keyboards available for pickup at my office
- Project 2: less than 2 weeks left (Deadline 5/2)
  - Will have demo sessions
- No class on 5/2 (finish the project!)
- Final exam: Thursday 5/5, 11:45pm – 2:45pm
- Project 2 + Final = 55%

Protocol Messages

There are 10 different protocol messages:

Category	Messages
Cache to Memory Requests	`ShReq, ExReq`
Memory to Cache Requests	`WbReq, InvReq, FlushReq`
Cache to Memory Responses	`WbRep(v), InvRep, FlushRep(v)`
Memory to Cache Responses	`ShRep(v), ExRep(v)`

Cache State Transitions (from invalid state)

No.	Current State	Handling Message	Next State	Dequeue Message?	Action
1	C-nothing	Load	C-pending	No	ShReq(id,Home,a)
2	C-nothing	Store	C-pending	No	ExReq(id,Home,a)
3	C-nothing	WbReq(a)	C-nothing	Yes	None
4	C-nothing	FlushReq(a)	C-nothing	Yes	None
5	C-nothing	InvReq(a)	C-nothing	Yes	None
6	C-nothing	ShRep (a)	C-shared	Yes	updates cache with prefetch data
7	C-nothing	ExRep (a)	C-exclusive	Yes	updates cache with data

Cache State Transitions (from shared state)

No.	Current State	Handling Message	Next State	Dequeue Message?	Action
8	C-shared	Load	C-shared	Yes	Reads cache
9	C-shared	WbReq(a)	C-shared	Yes	None
10	C-shared	FlushReq(a)	C-nothing	Yes	InvRep(id, Home, a)
11	C-shared	InvReq(a)	C-nothing	Yes	InvRep(id, Home, a)
12	C-shared	ExRep(a)	C-exclusive	Yes	None
13	C-shared	(Voluntary Invalidate)	C-nothing	N/A	InvRep(id, Home, a)

Cache State Transitions (from exclusive state)

No.	Current State	Handling Message	Next State	Dequeue Message?	Action
14	C-exclusive	Load	C-exclusive	Yes	reads cache
15	C-exclusive	Store	C-exclusive	Yes	writes cache
16	C-exclusive	WbReq(a)	C-shared	Yes	WbRep(id, Home, data(a))
17	C-exclusive	FlushReq(a)	C-nothing	Yes	FlushRep(id, Home, data(a))
18	C-exclusive	(Voluntary Writeback)	C-shared	N/A	WbRep(id, Home, data(a))
19	C-exclusive	(Voluntary Flush)	C-nothing	N/A	FlushRep(id, Home, data(a))

Cache Transitions (from pending)

No.	Current State	Handling Message	Next State	Dequeue Message?	Action
20	C-pending	WbReq(a)	C-pending	Yes	None
21	C-pending	FlushReq(a)	C-pending	Yes	None
22	C-pending	InvReq(a)	C-pending	Yes	None
23	C-pending	ShRep(a)	C-shared	Yes	updates cache with data
24	C-pending	ExRep(a)	C-exclusive	Yes	update cache with data

Home Directory State Transitions

No.	Current State	Message Received	Next State	Dequeue Message?	Action
1	R(dir) & (dir = ε)	ShReq(a)	R({id})	Yes	ShRep(Home, id, data(a))
2	R(dir) & (dir = ε)	ExReq(a)	W(id)	Yes	ExRep(Home, id, data(a))
3	R(dir) & (dir = ε)	(Voluntary Prefetch)	R({id})	N/A	ShRep(Home, id, data(a))
4	R(dir) & (id ∉ dir) & (dir ≠ ε)	ShReq(a)	R(dir + {id})	Yes	ShRep(Home, id, data(a))
5	R(dir) & (id ∉ dir) & (dir ≠ ε)	ExReq(a)	Tr(dir)	No	InvReq(Home, dir, a)
6	R(dir) & (id ∉ dir) & (dir ≠ ε)	(Voluntary Prefetch)	R(dir + {id})	N/A	ShRep(Home, id, data(a))

Messages sent from site *id*

Home Directory State Transitions

No.	Current State	Message Received	Next State	Dequeue Message?	Action
7	R(dir) & (dir = {id})	ShReq(a)	R(dir)	Yes	None
8	R(dir) & (dir = {id})	ExReq(a)	W(id)	Yes	ExRep(Home, id, data(a))
9	R(dir) & (dir = {id})	InvRep(a)	R(ε)	Yes	None
10	R(dir) & (id ∈ dir) & (dir ≠ {id})	ShReq(a)	R(dir)	Yes	None
11	R(dir) & (id ∈ dir) & (dir ≠ {id})	ExReq(a)	Tr(dir-{id})	No	InvReq(Home, dir - {id}, a)
12	R(dir) & (id ∈ dir) & (dir ≠ {id})	InvRep(a)	R(dir - {id})	Yes	None

Messages sent from site *id*

Home Directory State Transitions

No.	Current State	Message Received	Next State	Dequeue Message?	Action
13	W(id')	ShReq(a)	Tw(id')	No	WbReq(Home, id', a)
14	W(id')	ExReq(a)	Tw(id')	No	FlushReq(Home, id', a)
15	W(id)	ExReq(a)	W(id)	Yes	None
16	W(id)	WbRep(a)	R({id})	Yes	data -> memory
17	W(id)	FlushRep(a)	R(ε)	Yes	data -> memory

Messages sent from site *id*

Home Directory State Transitions

No.	Current State	Message Received	Next State	Dequeue Message?	Action
18	Tr(dir) & (id ∈ dir)	InvRep(a)	Tr(dir - {id})	Yes	None
19	Tr(dir) & (id ∉ dir)	InvRep(a)	Tr(dir)	Yes	None
20	Tw(id)	WbRep(a)	R({id})	Yes	data-> memory
21	Tw(id)	FlushRep(a)	R(ε)	Yes	data-> memory

Messages sent from site *id*

Acknowledgements

- These slides heavily contain material developed and copyright by
  - Krste Asanovic (MIT/UCB)
  - David Patterson (UCB)
- And also by:
  - Arvind (MIT)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
- MIT material derived from course 6.823
- UCB material derived from course CS252