Lecture 4: Update Protocol

- Topics: update protocol, evaluating coherence

Update Protocol (Dragon)

- 4-state write-back update protocol, first used in the Dragon multiprocessor (1984)
- Write-back update is not the same as write-through – on a write, only caches are updated, not memory
- Goal: writes may usually not be on the critical path, but subsequent reads may be

4 States

- No invalid state
- Modified and Exclusive-clean as before: used when there is a sole cached copy
- Shared-clean: potentially multiple caches have this block and main memory may or may not be up-to-date
- Shared-modified: potentially multiple caches have this block, main memory is not up-to-date, and this cache must update memory – only one block can be in Sm state
- In reality, one state would have sufficed – more states to reduce traffic

Design Issues

- If the update is also sent to main memory, the Sm state can be eliminated
- If all caches are informed when a block is evicted, the block can be moved from shared to M or E – this can help save future bus transactions
- The wire used to determine exclusivity is especially useful for an update protocol

Example

	P1			P2		
	MSI	MESI	Dragon	MSI	MESI	Dragon
• P1: Rd X						
• P1: Wr X						
• P2: Rd X						
• P1: Wr X						
• P1: Wr X						
• P2: Rd X						
• P2: Wr X						

Total transfers:

Evaluating Coherence Protocols

- There is no substitute for detailed simulation – high communication need not imply poor performance if the communication is off the critical path – for example, an update protocol almost always consumes more bandwidth, but can often yield better performance

- An easy (though, not entirely reliable) metric – simulate cache accesses and compute state transitions – each state transition corresponds to a fixed amount of interconnect traffic

State Transitions

From \ To	NP	I	E	S	M
NP	0	0	1.25	0.96	1.68
I	0.64	0	0	1.87	0.002
E	0.20	0	14.0	0.02	1.00
S	0.42	2.5	0	134.7	2.24
M	2.63	0.002	0	2.3	843.6

NP – Not Present

State transitions per 1000 data memory references for Ocean

From \ To	NP	I	E	S	M
NP	--	--	BusRd	BusRd	BusRdX
I	--	--	BusRd	BusRd	BusRdX
E	--	--	--	--	--
S	--	--	Not possible	--	BusUpgr
M	BusWB	BusWB	Not possible	BusWB	--

Bus actions for each state transition

Cache Misses

- Coherence misses: cache misses caused by sharing of data blocks – true (two different processes access the same word) and false (processes access different words in the same cache line)

- False coherence misses are zero if the block size equals the word size

- An upgrade from S to M is a new type of “cache miss” as it generates (inexpensive) bus traffic

Block Size

- For most programs, a larger block size increases the number of false coherence misses, but significantly reduces most other types of misses (because of locality) – a very large block size will finally increase conflict misses

- Large block sizes usually result in high bandwidth needs in spite of the lower miss rate

- Alleviating false sharing drawbacks of a large block size:
  - maintain state information at a finer granularity (in other words, prefetch multiple blocks on a miss)
  - delay write invalidations
  - reorganize data structures and decomposition

Update-Invalidate Trade-Offs

- The best performing protocol is a function of sharing patterns – are the sharers likely to read the newly updated value? Examples: locks, barriers
- Each variable in the program has a different sharing pattern – what can we do?
- Implement both protocols in hardware – let the programmer/hw select the protocol for each page/block
- For example: in the Dragon update protocol, maintain a counter for each block – an access sets the counter to MAX, while an update decrements it – if the counter reaches 0, the block is evicted

Title

- Bullet