ECE M116C / CS 151B: Week 9 Section

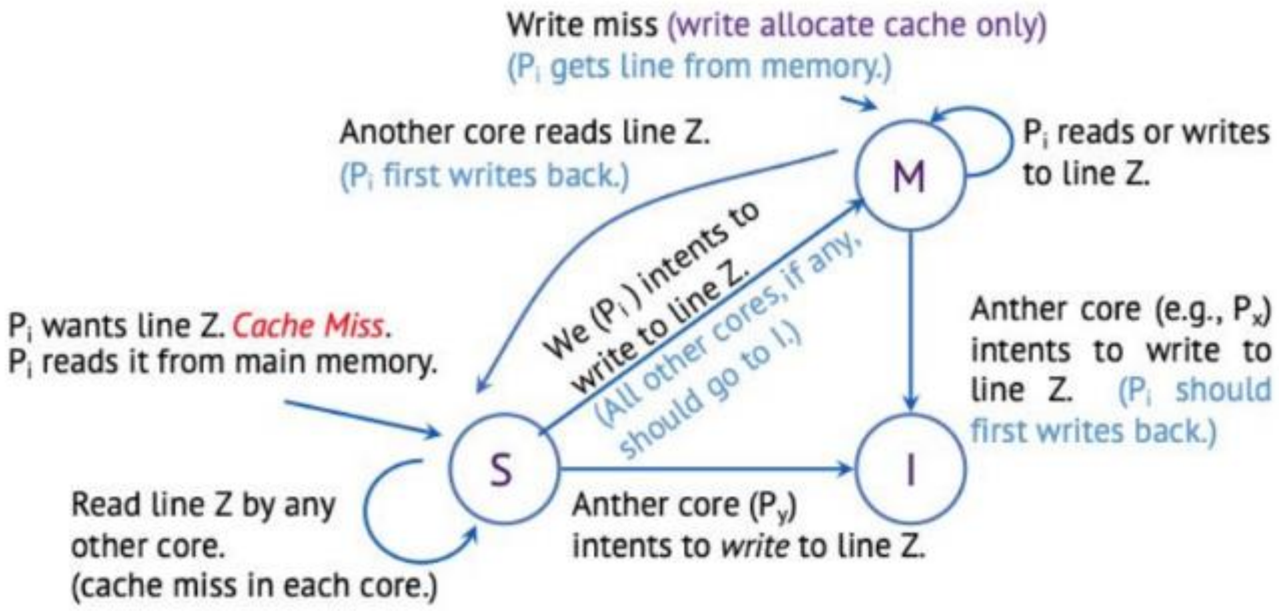
Justin Feng

Cache Coherency

- Need data to be uniform across different local caches for different cores (coherency is for **same address**!)
  - Parallel accesses of same address
- Snooping Cache: Memory bus between local caches and shared memory. Each unit broadcasts information to other units through the bus.

UCLA

Cache Coherency: MSI



Write miss (write allocate cache only)
(P_i gets line from memory.)
Another core reads line Z.
(P_i first writes back.)
P_i reads or writes to line Z.
M
We (P_i) intents to write to line Z.
(All other cores, if any, should go to I.)
P_i wants line Z. Cache Miss.
P_i reads it from main memory.
Anther core (e.g., P_x) intents to write to line Z. (P_i should first writes back.)
S
I
Read line Z by any other core. (cache miss in each core.)
Anther core (P_y) intents to write to line Z.

UCLA

Cache Coherency: MSI (Shared, modified, invalid)

- Shared: fresh copy of memory
- Modified: after a write. Now a “dirty” copy
- Invalid: if another core writes to same line, invalidate the current line
- Rules (using single address):
  - S: enter (read cache miss). Stay (read in other core). Go to M (write). Go to I (write in other core)
  - M: stay (read or write hit in current address. If write miss, get line from memory first). Go to I (write in other core). Go to S (read in other core). Note: if current core in “M”, and another core reads/writes to same address, we write back to memory first)
  - I: go to S (read cache miss)

UCLA

Cache Coherency: MSI Optimizations

- **MESI:** If current core has only copy, it is “exclusive” (“E”) – no need to broadcast after a write (go from “E” to “M” silently). If another core happens to read from same line, current core changes from “E” to “S”
- **MOSI:** If another core wants to read, and current core has newest copy (in “M”), current core becomes “owner” (“O”) and sends current copy to other core. No write back needed yet. If on “O”, or “M”, current core writes back and goes to “I” when another core writes.
- **MOESI:** both “E” and “O” optimizations at the same time
- **MOESIF:** instead of going to memory on a read, pass data from a forwarder “F”.

MSI Example

Processors P1, P2, P3

Read/Write sequence of same address:

- R1
- W3
- R2
- R1
- W2
- W1
- R3

	P1	P2	P3	WB?
	I	I	I	
R1				
W3				
R2				
R1				
W2				
W1				
R3				

MSI Example

Processors P1, P2, P3

Read/Write sequence of same address:

- R1
- W3
- R2
- R1
- W2
- W1
- R3

	P1	P2	P3	WB?
	I	I	I	
R1	S	I	I	
W3	I	I	M	
R2	I	S	S	P3: WB
R1	S	S	S	
W2	I	M	I	
W1	M	I	I	P2: WB
R3	S	I	S	P1: WB

UCLA

MOESIF Example

Processors P1, P2, P3

Read/Write sequence of same address:

- R1
- R3
- R2
- W2
- R1
- W3
- R1
- R2

	P1	P2	P3	WB?
R1				
R3				
R2				
W2				
R1				
W3				
R1				
R2				

UCLA

MOESIF Example

Processors P1, P2, P3

Read/Write sequence of same address:

- R1
- R3
- R2
- W2
- R1
- W3
- R1
- R2

	P1	P2	P3	WB?
	I	I	I	
R1	E	I	I	
R3	F	I	S	
R2	F	S	S	
W2	I	M	I	
R1	S	O	I	
W3	I	I	M	P2: WB
R1	S	I	O	
R2	S	S	O	

Cache Consistency

- Different from cache coherency - cache consistency is about parallel accesses **between different addresses**
- Sequential consistency:
  - See loads/stores between all cores in sequential order. Inefficient!
- Relaxed memory consistency:
  - Allow certain load/store orders to be violated
  - Total Store Order Model: stores executed by a singular processor kept in order. Use store buffers to hide latency of making store visible to other processors. Can then process reads quicker instead of waiting for visibility.
- Fences/Mutex/Atomic Instructions: tools used to maintain sanity in a weak-memory model

LSQ: Hazards

- **WAW**: store queue ensures stores are committed in order
- **WAR**: LW -> SW. If SW is run before LW, SW doesn’t go set memory immediately. LW grabs from memory, then in the LSQ, LW commits first, then SW commits.
- **RAW**: SW -> LW. If we speculatively run LW before SW, and SW happens to edit the same address, then we need to flush the instructions after LW, then rerun LW. Since we used LSQ, the other threads don’t see this.

TSO vs Weak Memory Model

- TSO: guaranteed to have LSQ. All stores are committed in order (seen in order by other threads). Prevents WAR and WAW and RAW hazards
- Weak Memory Model: may or may not have LSQ. If no LSQ: any order is possible (excluding control flow dependencies). If LSQ: prevents WAR and WAW and RAW hazards.

Fence vs Barrier vs Lock

- **Fence**: waits for all memory instructions before it to finish (per core)
- **Barrier (external fence)**: blocks execution of threads until the correct number of threads have reached a given point.
- **Lock**: prevents other threads from accessing a shared region of memory