Advanced Computer Architecture

Memory Ordering

Fall 2016

Pejman Lotfi-Kamran

Adapted from slides originally developed by Profs. Hill, Hoe, Falsafi and Wenisch of CMU, EPFL, Michigan, Wisconsin

Where Are We?

Fr	Sa	Su	Mo	Tu
	27-Shahrivar		29-Shahrivar	
	3-Mehr		5-Mehr	
	10-Mehr		12-Mehr	
	17-Mehr		19-Mehr	
	24-Mehr		26-Mehr	
	1-Aban		3-Aban	
	8-Aban		10-Aban	
	15-Aban		17-Aban	
	22-Aban		24-Aban	
	29-Aban		1-Azar	
	6-Azar		8-Azar	
	13-Azar		15-Azar	
	20-Azar		22-Azar	
	27-Azar		29-Azar	
	4-Dey		6-Dey	

- This Lecture
  - Consistency (2)
- Next Lecture:
  - Consistency (3)

The Big Debate

When shall hardware enforce order?

Always order:

- Always (SC)
  - ▲Software wants a "multiprogrammed CPU" behavior
- Easy to program & understand
- Assumed to exhibit inferior performance

Relax order:

- Order enforced through software annotation
- Let the hardware overlap write latency

Big Misconception: Large Performance Gap

But, strong ordering thought to hurt performance

- One reason for a variety of memory models and flavors

Not true!

Memory only has to appear to be ordered

- Hardware can relax order speculatively
- Save state while speculating
- Roll back if relaxed order observed by others
- E.g. result, SC + Speculation ≥ RC!

This is the Bart Simpson's approach to relaxing order:
"I didn't do it. Noone saw me doing it!"

Evolution of SC Systems

Naiive SC: every access is ordered

Optimized SC:
- Three simple optimizations
- Existing pipeline HW
- E.g., MIPS R10K

Wait-free SC:
- SC++
- ASO
- InvisiFence

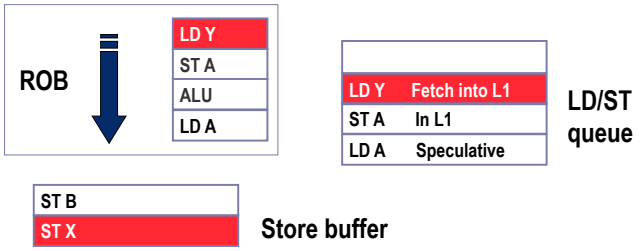
Enhancing SC's Performance

1. Review: Store buffering
   - Store misses (either missing data or permission) can be removed from pipeline in program order
   - Place them in a store buffer
2. Review: All miss "fetches" can be overlapped
   - L1 cache is a point of serialization (think "switch")
   - Requests for cache blocks can be sent out in parallel
   - All accesses to L1 must be atomic and in order
   - Therefore:
     - Order in which blocks are fetched/filled has no bearing on actual observed program order
     - Not until L1 cache access for the block is performed
3. New: Load hits can "speculate" past store misses
   - **MIPS R10000 did this!**

Review: SC + Store Buffer + Fetch blocks

```
ST X    ; removed from pipeline (wait in SB)
ST B    ; removed from pipeline (wait in SB)
LD A    ; looks up cache but waits (no load)
ALU     ; waiting for LD A
ST A    ; looks up cache but waits (no store)
LD Y    ; looks up cache, fetches Y into L1
ST Z    ; instruction not fetched
```


SC + Store Buffer + Fetch + Speculative Load



Loads A speculatively [Gharachorloo' 91,Ranganathan' 97]

+ LD A, ALU and ST A are done

Not enough buffering (ROB full)!

MIPS R10K: SC + Store Buffer + Fetch + Speculative Load

```
ST X    ; removed from pipeline (wait in SB)
ST B    ; removed from pipeline (wait in SB)
LD A    ; speculatively loaded
ALU     ; Done!
ST A    ; waiting for store buffer
LD Y    ; looks up cache, fetches Y into L1
ST Z    ; instruction not fetched
```


What if another processor asks for A?

LD A is out of order with respect to ST X & ST B!

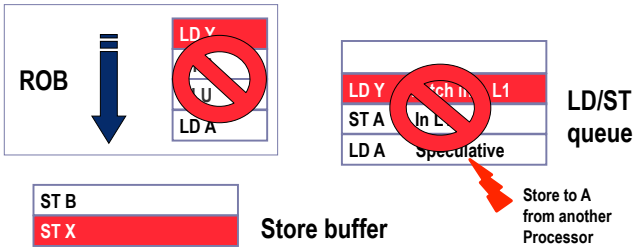
What if another processor stores A meanwhile?

- Coherence message sent to this processor
- LD/ST queue has a speculative bit for LD A
- If the bit is set ➔ **misspeculation!**

On misspeculation:

- Service the coherence message (e.g., invalidate A)
- Re-execute starting from the LD A
- LD A will turn into a miss
- Noone saw the reordering (except for the local processor)!

Misspeculation in MIPS R10K



Store to A from another processor

- Flush all ROB and LD/ST queue entries
- Re-execute from LD A

Requirements for Store-Stall-Free SC: SC++ [Gniady'99]

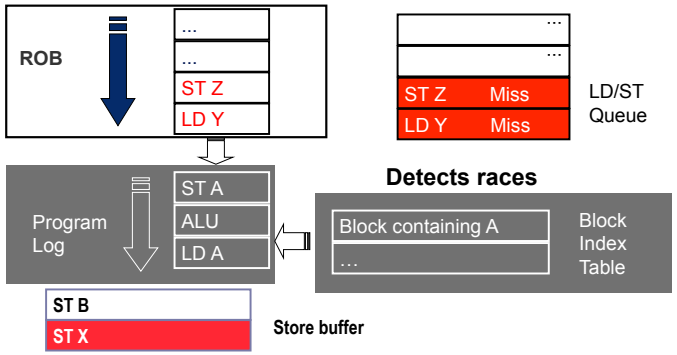
① Non-blocking caches

② Logging program state while a store is pending
- May need to log thousands of cycles

③ Fast lookup to detect order violation
- Upon request for data by others

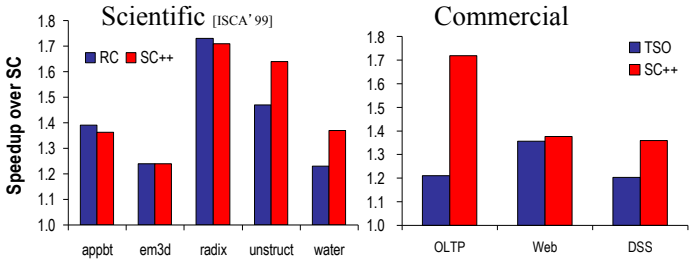
④ Infrequent rollbacks
- Typical of well-behaved parallel applications
- Rollbacks are due to false sharing or data races

SC++: A Store-Stall-Free SC



- Maintain a "log" of computation, with a fast block lookup
- All order relaxed up to first missing LD (LD Y)!

Performance Comparison



- Data from RSIM (left) and Flexus (right)
  - 16 1-GHz OoO processors, Piranha DSM
  - TSO = Stores can not bypass each other
- Up to 70% gap between SC & SC++
- SC++ > RC >> TSO

Can we do any model ++?

- Relaxed models conservatively enforce order at Fences!
  - If there are may fences in the code performance hurts
  - E.g., XCHG instructions in PC (TSO)
  - E.g., Fence instructions in WC & RC
- PC++, WC++, RC++?
  - Yes!
  - Can use the same buffering and lookup
  - Relax Fence instructions!

Summary

- Can relax order as long as
  - There is enough buffering
  - The buffer can be looked up for races (coherence messages)
- Relaxed models may not be necessary to achieve high performance!
  - Hardware can relax order
  - Models may be needed to allow compilers reorder
  - Compilers have not had much impact on performance