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 (3)
- Next Lecture:
  - Synchronization

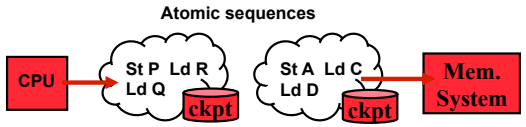
Atomic sequence ordering (ASO) [Wenisch' 07]

- Intuition: If access sequence appears atomic, then actual order does not matter
- Dynamically group accesses into atomic sequences
- Relax order within & across sequences
- Stores: get all permissions; then commit atomically
  - On-chip commit (into L2)
- On a race, recover to sequence's checkpoint
  - Races are rare in typical parallel apps. [Gniady' 99]

Coarse-grain rollback enables practical HW

Atomic sequence ordering (ASO)

- Enforce order over coarse-grain atomic sequences

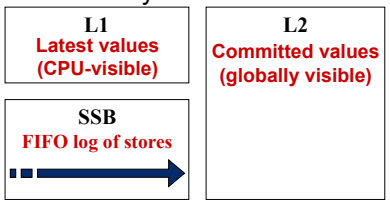


- Detect races:
  - Mark speculative loads in cache
  - Coherence request to speculative block ⇒ violation
- CPU rollback: 2-4 CPU checkpoints

Coarse-grain rollback enables practical HW

Scalable Store Buffer (SSB)

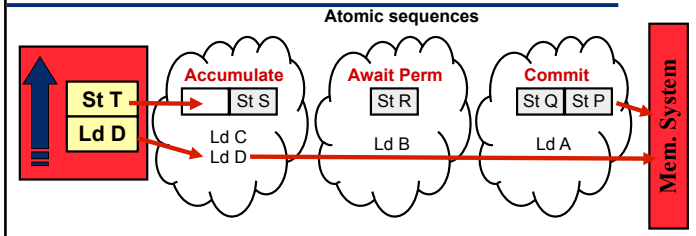
- Maintain consistency at L2 instead of L1



- Fast access to latest value in L1
- Commit consistent values from SSB → L2

SSB is FIFO – no RMW, no associative search

Operation under ASO



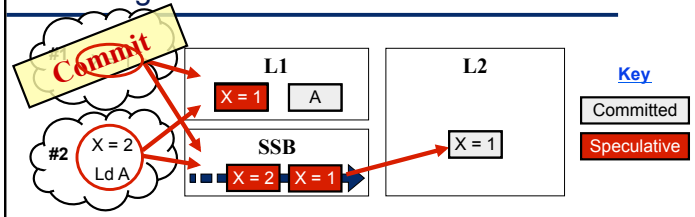
Accumulate accesses into sequence

Await permission for all stores

Commit writes atomically

Obtain permissions in advance → on-chip commit

Handling stores: Commit

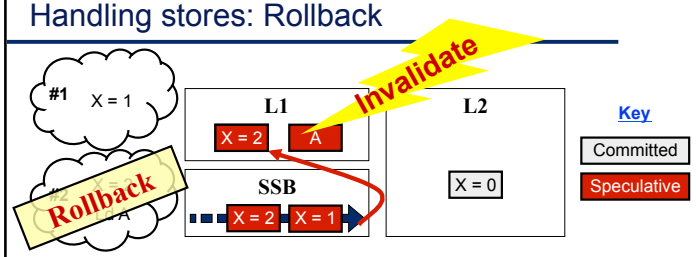


Store to L1 and append to SSB

Drain updates from SSB to L2

Delay external requests while draining for atomicity

Handling stores: Rollback



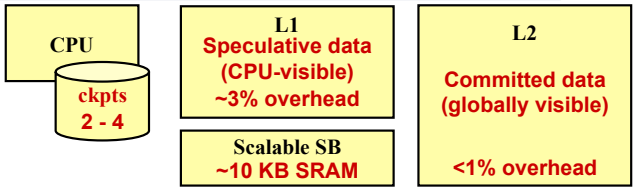
Detect race on external request

Discard all speculative blocks from L1

Replay stores SSB → L1 (similar to [Gandhi 05])

Forward progress: 1 non-spec. store if 1st seq. rolls back

ASO Hardware



- Fast access to CPU-visible and globally-visible values
  - Stores: issue to L1&SSB; commit SSB→L2
- Detect races: L1 & L2 track sequences' reads
  - Recover to checkpoint; reconstruct L1 from SSB
- Similar HW for TLS, TM → add generic mechanisms

Evaluation methodology

- Flexus OoO timing simulation
- SMARTS statistical sampling

Benchmark Applications

OLTP: TPC-C
- IBM DB2 & Oracle

DSS: TPC-H Qry 1, 6, 16
- IBM DB2

Web: SPECweb99
- Apache & Zeus

Scientific
- barnes, ocean

Model Parameters

16-node directory-based DSM

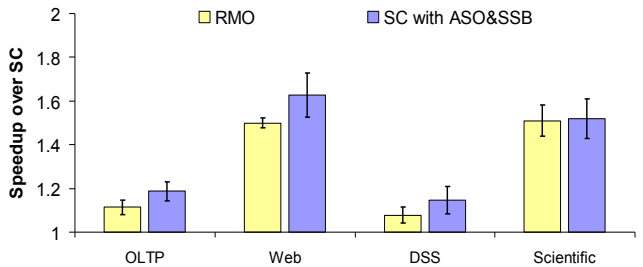
4GHz CPUs; SPARC ISA

4-wide OoO; 8-stage pipeline

64KB L1, 8MB L2

Store prefetch & speculative loads [Gharachorloo' 91]

Performance comparison



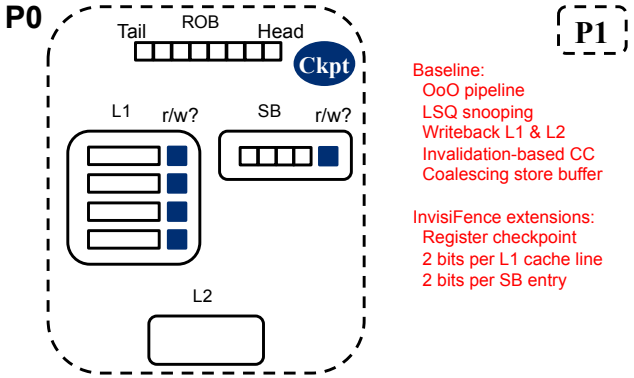
- Up to 50% gap between aggressive SC and RMO
- Even under RMO, fences cause 5-10% unnecessary stalls

ASO & SSB enable ≥RMO perf. with SC guarantees

InvisiFence For Weak Ordering

- Add deep speculation to eliminate stalling on fences
- Mechanism: register ckpt + 2 bits per L1 cache line
  - Similar HW to other deep speculation (TLS, TM, Cherry…)
- Initiate speculation at fence instructions
  - Detect violations via cache coherence protocol
  - Preserve non-speculative data in L2 (facilitates rollback)
- Speculation ends when store buffer becomes empty
  - Commit by flash-clearing read/write bits

InvisiFence Hardware

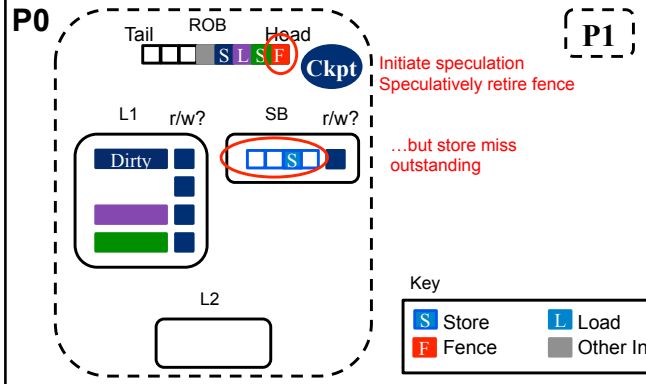


Baseline:
- OoO pipeline
- LSQ snooping
- Writeback L1 & L2
- Invalidation-based CC
- Coalescing store buffer

InvisiFence extensions:
- Register checkpoint
- 2 bits per L1 cache line
- 2 bits per SB entry

InvisiFence: Example

Fence wants to retire…

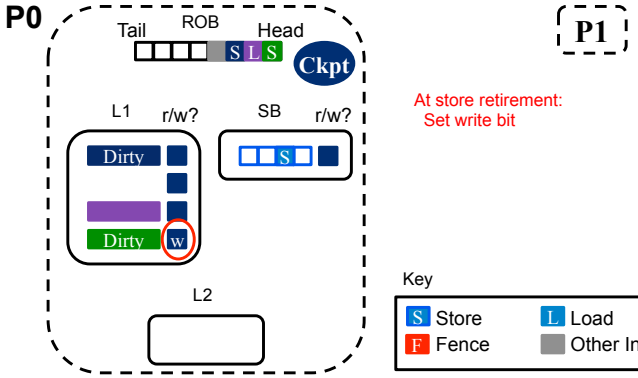


Initiate speculation
Speculatively retire fence

…but store miss outstanding

Key
- S Store
- L Load
- F Fence
- Other Insn

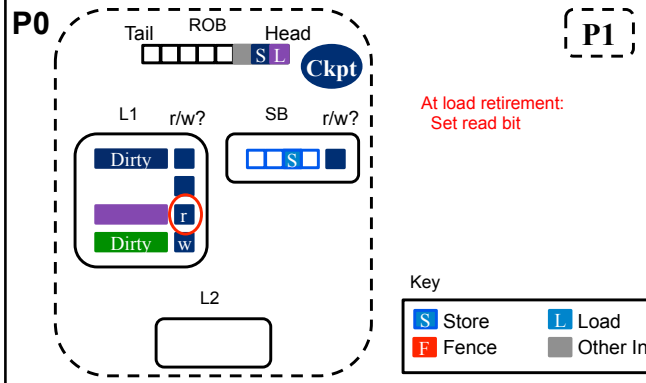
InvisiFence: Violation Detection



At store retirement:
Set write bit

Key
- S Store
- L Load
- F Fence
- Other Insn

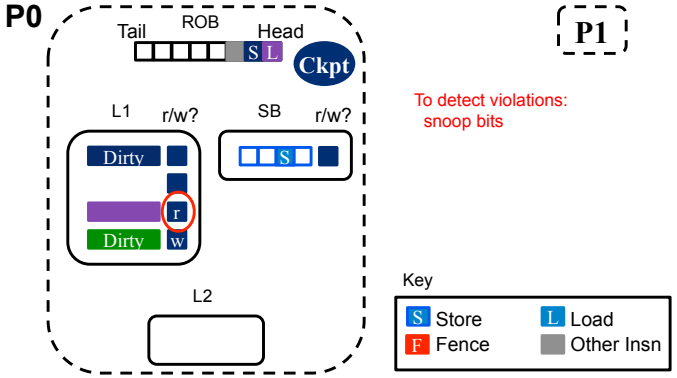
InvisiFence: Violation Detection



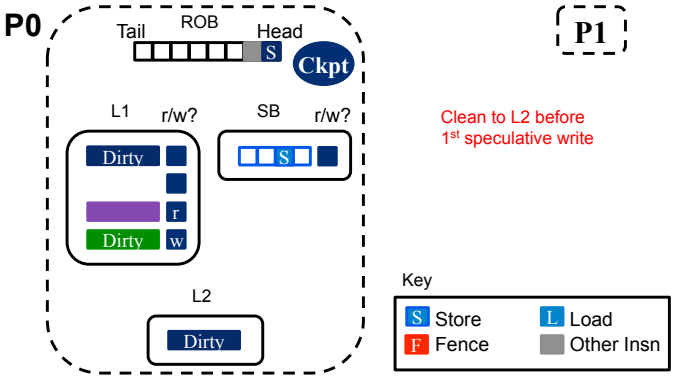
At load retirement:
Set read bit

Key
- S Store
- L Load
- F Fence
- Other Insn

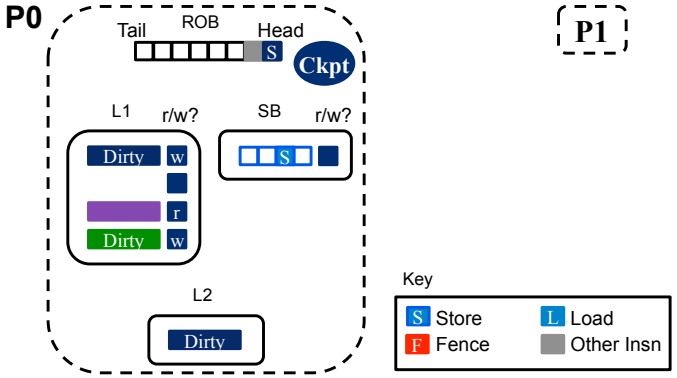
InvisiFence: Violation Detection



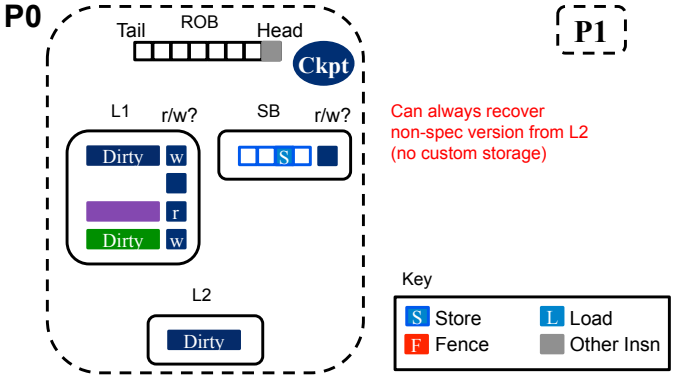
InvisiFence: Version Management



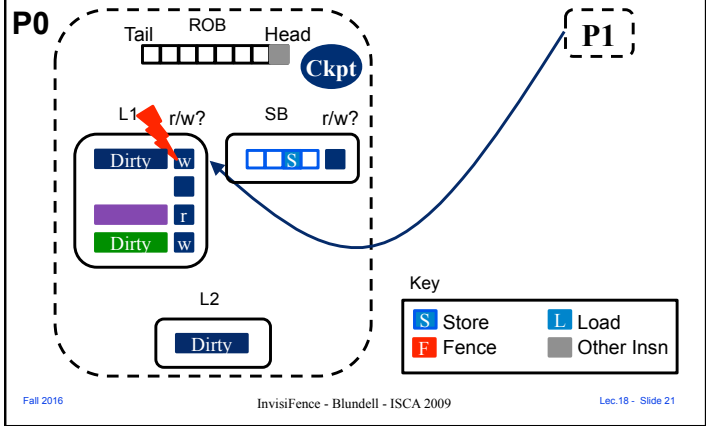
InvisiFence: Version Management



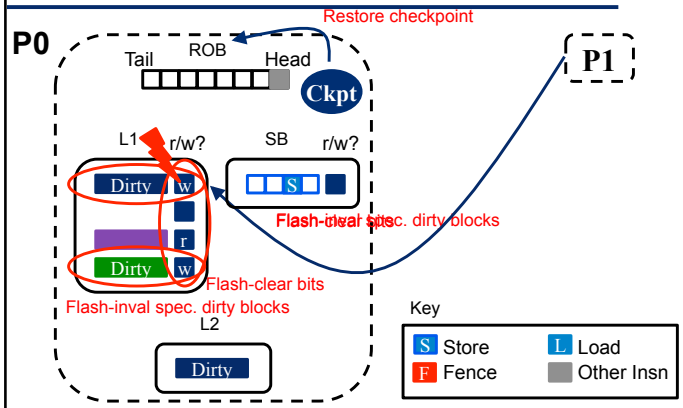
InvisiFence: Version Management



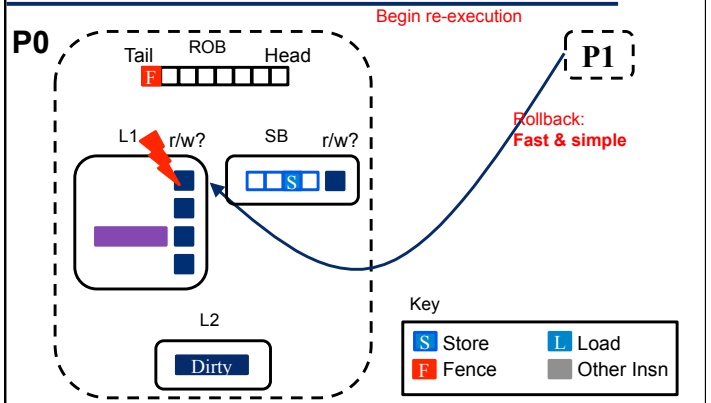
InvisiFence: Rollback



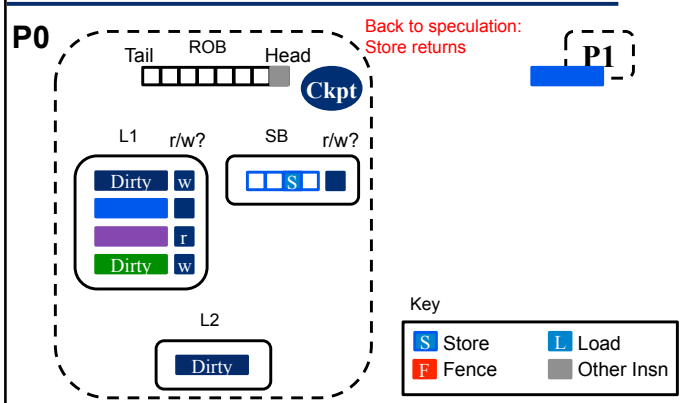
InvisiFence: Rollback



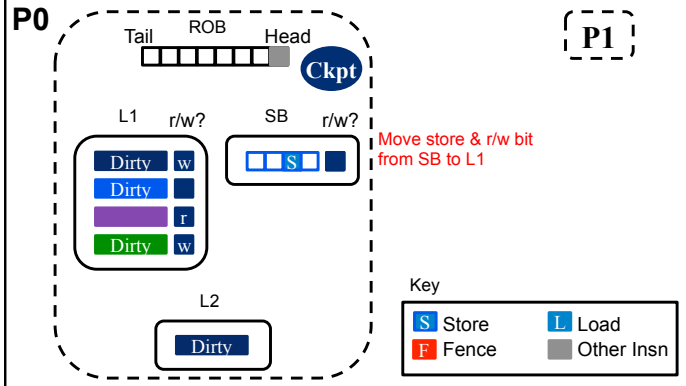
InvisiFence: Rollback



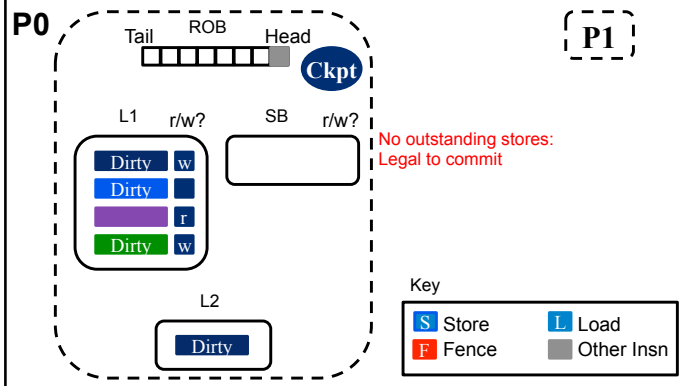
InvisiFence: When to Commit?



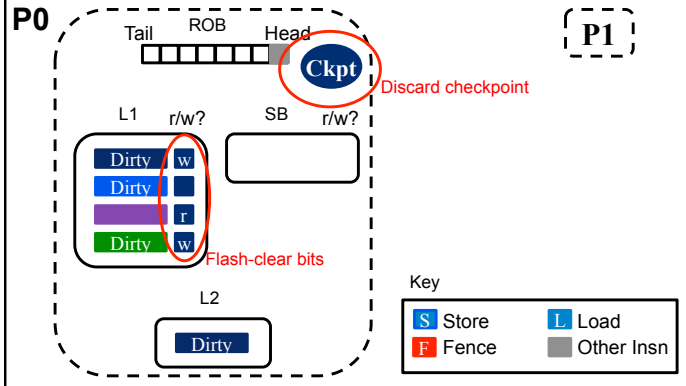
InvisiFence: When to Commit?



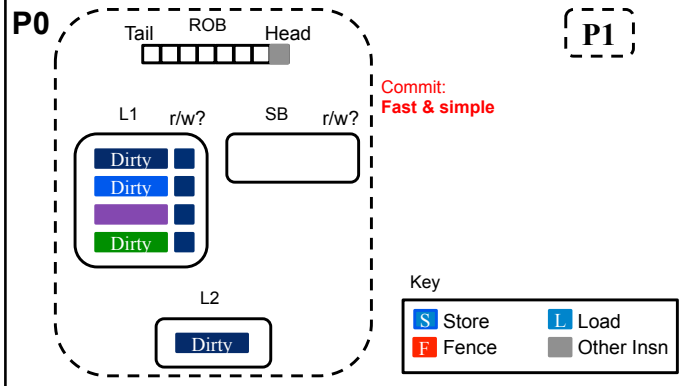
InvisiFence: When to Commit?



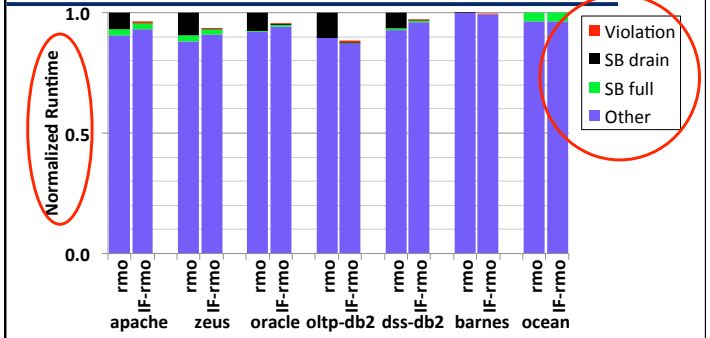
InvisiFence: Commit



InvisiFence: Commit



InvisiFence Performance

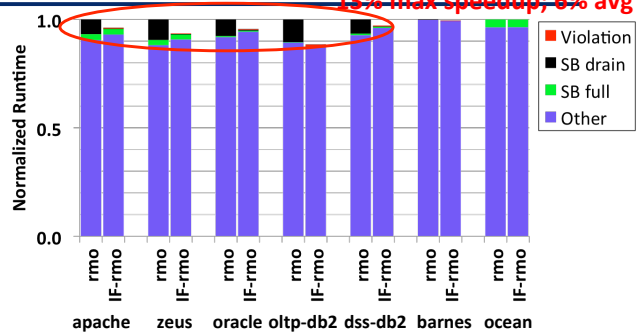


Flexus simulation of 16-node directory-based SPARC MP

SPARC's RMO (similar to Alpha, ARM, PowerPC)

InvisiFence Performance

13% max speedup; 6% avg



InvisiFence eliminates fence stalls without violations

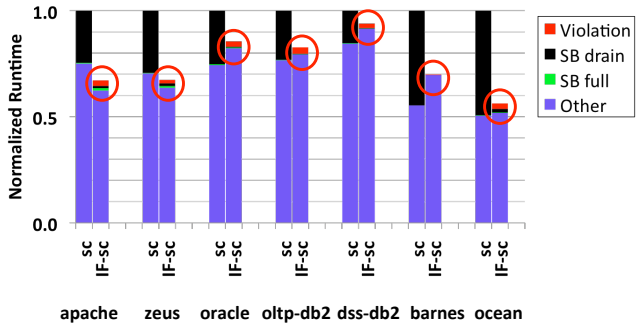
But what about models requiring stronger ordering?

Generalizing InvisiFence for Strong Ordering

- Strong models impose additional ordering constraints
  - Processor Consistency (x86, TSO): ordering between stores
  - Sequential Consistency: ordering between all operations
- These constraints are conceptually "implicit fences"
  - *e.g.*, for SC: every operation is "implicit fence"
- InvisiFence can handle these just like explicit fences!
  - Increases speculation frequency…

No other hardware changes

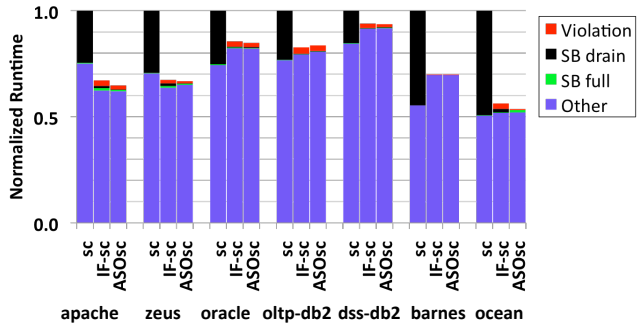
Strong Ordering Performance (SC)



Violations are negligible (3% slowdown from IF-RMO)

How does this compare to prior work?

Strong Ordering Performance (SC)



Comparison to Atomic Sequence Ordering [Wenisch`07]: Both eliminate stalls

ASO & InvisiFence: Design Comparison

ASO [Wenisch'07]

- Fine-grained tracking
  - 1K-entry store buffer
  - 10 KB
- Lengthy commit
  - Atomically drain SB to L2
  - Multiple checkpoints
- Changes to L1
  - Mult. per-block R/W bits
  - Write-through
  - Per-word valid bits

InvisiFence

- Coalesced tracking
  - 8-entry store buffer
  - < 1 KB
- Constant-time commit
  - Flash-clear bits
  - Single checkpoint
- Changes to L1
  - Single per-block R/W bits
  - Clean to L2

Both eliminate stalls, but InvisiFence hardware simpler

Summary: Speculative Memory Ordering

Memory Consistency models are neither necessary nor sufficient to achieve high performance!

- No need for "static" compile-time fences
- Detect and enforce fences dynamically
- Speculation techniques effective
- Similar mechanisms needed for Transactional Memory or Lock speculation