SCsafe: Logging Sequential Consistency Violations Continuously and Precisely

Yuelu Duan, David Koufaty¹, and Josep Torrellas University of Illinois at Urbana-Champaign http://iacoma.cs.uiuc.edu

¹Intel Labs



HPCA March 2016



Sequential Consistency (SC)



- In SC, memory accesses:
 - Appear atomic
 - Have a total global order
 - For each thread, follow program order





Sequential Consistency Violation (SCV)

- SCV: access reorder that does not conform to SC
- Machines support relaxed models, not SC
- Machines may induce SC violations (SCV)







When Can an SCV Occur?

- Two or more data races overlap
- They create a cycle







- Programmers assume SC
 - SCV is almost always a bug: unexpected interleaving
 - Single-stepping debuggers cannot reproduce the bug
- Causes portability problems
 - Code may not work across machines
- Traditional data race detectors won't work to find SCVs
 - Not specific enough
 - Some codes use races intentionally





Contribution: SCsafe

- First architecture that detects and logs SCVs continuously
 - Records SCV
 - Recovers execution and continues transparently
 - Retains SC
- Compatible with production runs: does not crash
- Finds true SCVs; to be fixed later
- Precise: no false alarms due to false sharing
- Modest hardware support
- In codes with few SCVs, negligible performance overhead





Current Approaches are Insufficient

- Only enforce SC
 - Look for a necessary condition for SC: observe a speculative access
 - Squash thread



 Conservative: cycle may never happen

- Detect one SCV and then stop
 - Detect cycle by passing time-stamps



i-acoma group

- After detection, program is not SC → program has to terminate
- Hardware is complicated



M-Speculative == "speculative relative to the memory model of the processor" Its an access that

- Is reordered AND
- If it is observed, it will be squashed



- In TSO: rd(y) is M-speculative: it will be squashed
- In RC: rd(y) is not M-speculative: it will not be squashed

We are interested in accesses that are NOT M-Speculative





SCsafe Idea (I)

- HW keeps track of a processor's accesses that are reordered AND not M-speculative
 - Would not be squashed if observed
- HW *nacks* any incoming coherence transaction directed to addresses of these accesses



• HW stops *nacking* when access is not reordered anymore





9

SCsafe Idea (II)

- When we have a nack cycle: two or more cores enter deadlock
 - An SCV has been prevented from happening



• SCsafe detects the deadlock

– Logs the SCV: addresses + PCs

- SCsafe forces at least one thread to rollback the reordered accesses and re-execute them
- Execution continues at production-run speeds
- SC is retained \rightarrow future SCVs are real SCVs





- Key idea: Never satisfy a request that may end up closing a dependence cycle; stall it instead
 - No need for timestamps to identify cycles, unlike past schemes
 - Simply look for a deadlock
- No incorrect data has been supplied
 - Easy to rollback
 - Rollback only one thread, and correct execution can resume
- Need to ensure that reordered accesses can be undone
 - Reordered stores perform an exclusive prefetch, not a write





Architecture Support





Architecture Support: Reordered Set (RS)

- Queue in the cache controller
- Keeps addresses of reordered, non M-speculative accesses
- Checked on incoming coherence transactions: nacks if conflict
- Accesses removed when they are not reordered any more





Architecture Support: **Deadlock Detector (DD)**

- FSM triggered when:
 - The core nacks an external request, AND
 - The oldest request by the core is nacked by another core
- Then, the retry messages are augmented with a core bitmap
- Each core in the deadlock sets a bit in the bitmap. See paper





Architecture Support: History Buffer (HB)

- Contains "undo" state of each reordered retired instruction
- As a reordering terminates, HB entries freed
- In a deadlock, cores have executed reordered accesses
 - Memory not polluted (reordered stores only do exclusive prefetch)
- To recover: use HB to undo the reordered instructions of 1 core





Types of Stalls



Some go away

3-way cycles





Detect, do not record SCV, recover, and resume





- Simulations of 16-core multicore. Cores are 3-issue ooo
- Workloads:
 - 12 small programs that implement concurrency algorithms
 - Fences are removed, and hence may have SCVs
 - Goal: measure SCsafe's ability to find SCVs
 - 16 SPLASH-2 and PARSEC
 - No SCVs (although false-sharing induced cycles)
 - Goal: measure the execution overhead
 - Compare overhead to *InvisiFence*: SC-enforcement only (squash when reordered access is observed)





SCsafe Detects and Records SCVs

Program	RC		TSO	
	# of SCVs	# of Stalls	# of SCVs	# of Stalls
Bakery	3	4494	3	4362
Dekker	14	91412	17	83093
Harris	302	23256	191	24010
Average	110	17188	66	16147

- SCsafe detects many SCVs
- Most of the stalls do not result in deadlocks





18

SCsafe Execution Overhead over RC No Checks



- SCsafe has very small overhead: 2% average over RC no checks
- SCsafe as fast as InvisiFence, which only supports SC enforcement (squash when SCV possible), does not log SCVs





Also in the Paper

- Rigorous definition of the terms used
- Detailed explanation:
 - Deadlock detection and recovery algorithm
 - Operation of the Reorder Set and History Buffer
- Livelock considerations
- Hardware complexity
- Extensive evaluation





Conclusions

- SCsafe: First architecture that detects and logs SCVs continuously
 - Logs SCV
 - Recovers and continues execution
 - Retains SC
- Compatible with production runs: does not crash
- Finds true SCVs; to be fixed later •
- Precise: no false alarms due to false sharing ٠
- Modest hardware support \bullet
- In codes with few SCVs, negligible performance overhead (2%) ٠





SCsafe: Logging Sequential Consistency Violations Continuously and Precisely

Yuelu Duan, David Koufaty¹, and Josep Torrellas University of Illinois at Urbana-Champaign http://iacoma.cs.uiuc.edu

¹Intel Labs



HPCA March 2016









SCsafe Execution Overhead over RC No Checks



- SCsafe has very small overhead: 2% average over RC no checks
- SCsafe as fast as InvisiFence, which only supports SC enforcement (squash when SCV possible), does not record SCVs



