Paper Reading 2 of CS211 in Fall 2022

Anonymous

1 Paper

A. Ros and S. Kaxiras, "Speculative Enforcement of Store Atomicity," 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2020, pp. 555-567, doi: 10.1109/MICRO50266.2020.00053.

2 Problem and Theory Solution

Model	**Adve & Gharachorloo**	**Trippel et al.**	**Ros & Kaxiras**
370		MCA	Store atomicity
x86	Read own write early	rMCA	Write atomicity
PC	Read others' write early	non-MCA	Non write-atomic

These are current memory consistency model implementations, X86-TSO: a core see its own stores before other cores, while IBM 370 all cores see all stores atomically, x86-TSO is less intuitive than IBM 370, the author want to have a more stricter, more readable and more intuitive solution.
The author believed that a load performed by a forwarding from an $in-limbo$ store is not speculative, younger loads performed after that forwarding are. The author put up with a idea that they only perform dynamically enforcement of store atomicity only when the detection of its violation occurs.

3 Background of X86 Non-store-atmoic Semantics

3.1 Ordered Stores

Load a value will not be performed until store the same value is inserted in the memory order and all cores can see the new value, preventing loads seeing different store orders.

3.2 Independent Stores

Load value x in one core and load value y in another core will not be able perform until the respective stores in the two cores are written from the store buffers to the L1s and hence inserted in the memory order. Here is three possible situations and one impossible situation.

Case	Core1 [x],[y]	Core2 [x],[y]	Comment
1	1,0 (new,old)	0,1 (old,new)	Disagreement in order
2	1,0 (new,old)	1,1 (new,new)	Core2 cannot see order
3	1,1 (new,new)	1,0 (new,old)	Core1 cannot see order
4	1,1 (new,new)	1,1 (new,new)	None can see any order

If one of the cores is able to discern an order between the two independent stores, then another core cannot, that is, there is no disagreement about the order of independent stores in a store-atomic implementation.

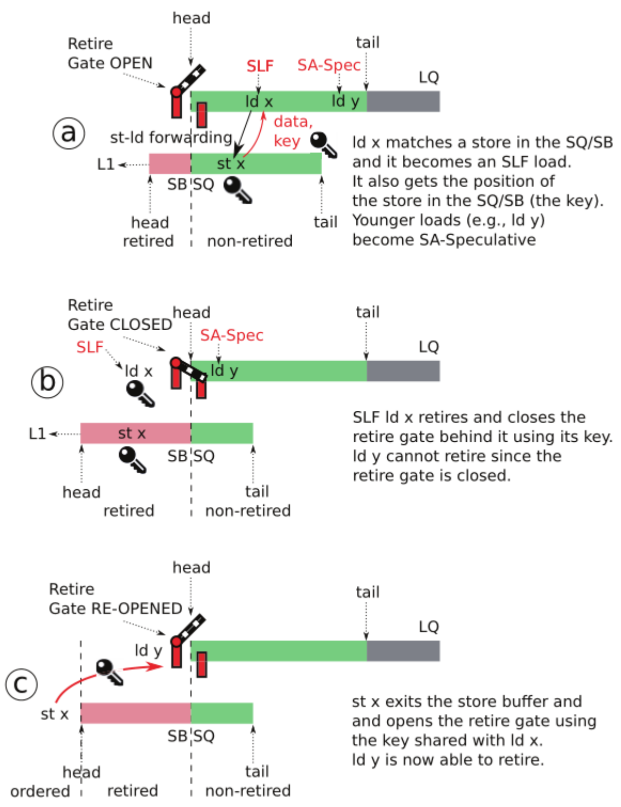
4 Speculative Enforcement of Store Atomicity

Two solutions to guarantee store atomicity are

- Prevent the consumer load of the store-to-load forwarding, from being performed until store is inserted in memory order.

- Treat load as speculative, not allowing it to retire until the store buffer empties.

They both highly effect performance.
The author will make load value y in core 1 speculative until store x is ordered. If an invalidation is received for y in the interim, it must squash load y and re-excute it. The author's implementation is base on SLF loads, that is, an SLF(Store-to-Load-Forwarded) load establishes a connection between a store in the store buffer and a "retire gate" at the head of the load queue, to prevent the retirement of any younger SA-speculative loads until the store is inserted in memory order.



ld x matches a store in the SQ/SB and it becomes an SLF load. It also gets the position of the store in the SQ/SB (the key). Younger loads (e.g., ld y) become SA-Speculative

SLF ld x retires and closes the retire gate behind it using its key. ld y cannot retire since the retire gate is closed.

st x exits the store buffer and and opens the retire gate using the key shared with ld x. ld y is now able to retire.

- Step 1: Setting SLF loads on store-to-load forwarding, the load receives its value and is marked as an SLF and keeps a pointer to the position of the store in the SB.
- Step 2: Closing the retire gate, if the load has the valid key of the older store, the retire gate is closed by the key of the load, preventing younger SA-speculative loads from retiring.
- Step 3: Reopening the retire gate, if the key of the store matches the key(closed the door), then the retire gate is reopened.

In summary, By marking the load queue and prevent load operations to commit, it can stall loads in the reorder buffer until the store that forwards value to load commit.

5 Evaluation Environment

Sniper with in-house processor model and GEMS

- Reorder buffer: 224 entries
- SB: 56 entries
- Load queue: 72 entries
- Processor: 8 out-of-order Intel Skylake cores

For parallel benchmark, using Splash-3 and Parsec-3.0.
For sequential benchmark, using SPEC CPU 2017.

6 Results and Conclusion

The author get an efficient implementation of the IBM 370 model with similar performance(+2.7% performance) (<3% overhead) than x86-TSO at a low complexity and hardware cost.

7 Thinking and Questions

- For multiple store-forward-loads on different addresses, is it possible for each SFL start a new gate or the later loads is treated as speculative?
- Since it do not have very significant performance improvement compared with stalling the forwarding load, it might caused by overlappings between the draining of the store buffer and the execution of the next load.