TOSTING: Investigating Total Store Ordering on ARM

ARCS'23

Lars Wrenger Dominik Töllner Daniel Lohmann

Leibniz Universität Hannover
{wrenger,toellner,lohmann}@sra.uni-hannover.de

14.06.23

Motivation - Memory Models

- Backward compatibility with Rosetta 2 (x86 translation layer)
- Emulation is challenging due to different **Memory Ordering** models

Memory Ordering

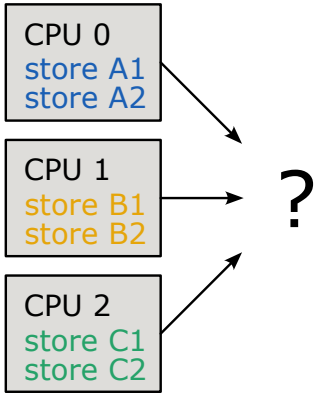
- CPUs execute instructions in parallel and out of order to hide latencies

Memory Ordering – Gharachorloo et al. 1990; Higham et al. 1997; Pulte et al. 2017

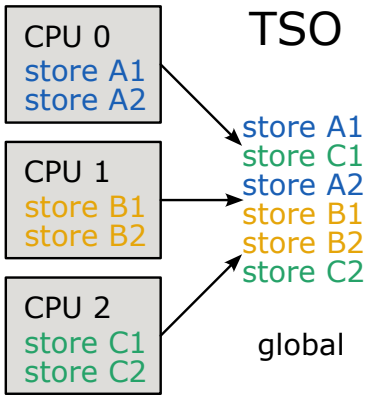
Defines how accesses to shared memory can be observed by other participants.

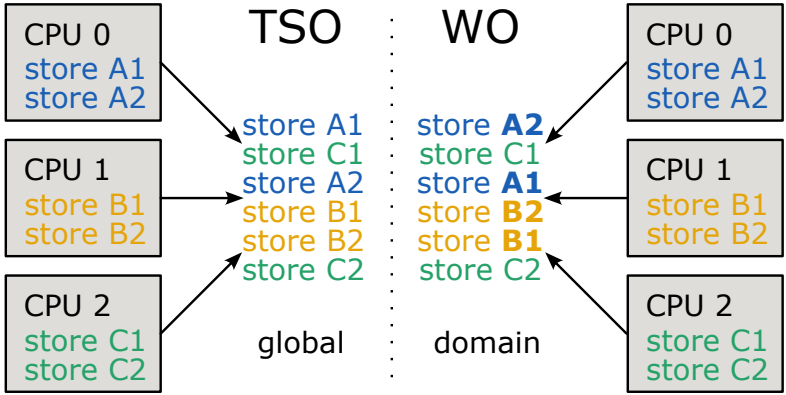
- Stricter ordering simplifies the programming model
  - Total Store Ordering (TSO) on x86
- Weaker ordering allows more latency hiding
  - Weak Ordering (WO) on ARMv8

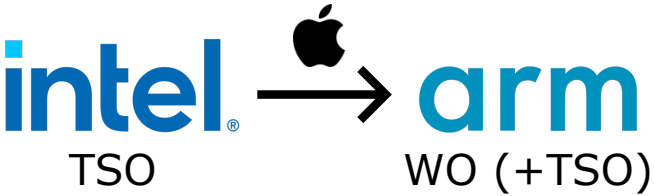
Memory Ordering



Memory Ordering

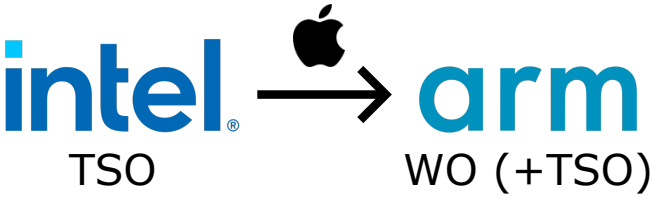



CPU 0
store A1
store A2
CPU 1
store B1
store B2
CPU 2
store C1
store C2
TSO
store A1
store C1
store A2
store B1
store B2
store C2
global
WO
store A2
store C1
store A1
store B2
store B1
store C2
domain
CPU 0
store A1
store A2
CPU 1
store B1
store B2
CPU 2
store C1
store C2



- Emulating TSO on a WO platform requires synchronizing every memory access.
- The Apple M1 implements TSO in hardware for the x86 emulation

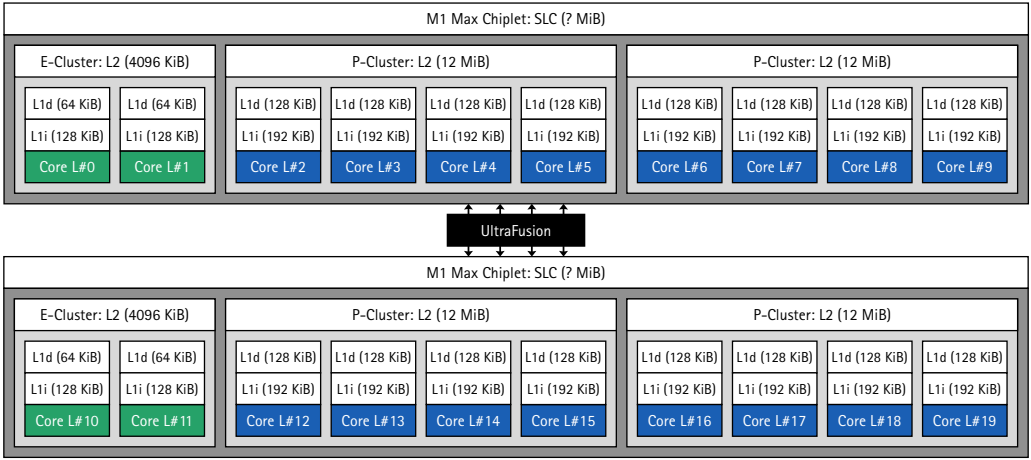
Memory Ordering



- Emulating TSO on a WO platform requires synchronizing every memory access.
- The Apple M1 implements TSO in hardware for the x86 emulation

What impact do different memory ordering models have on performance?

The Apple M1 Ultra

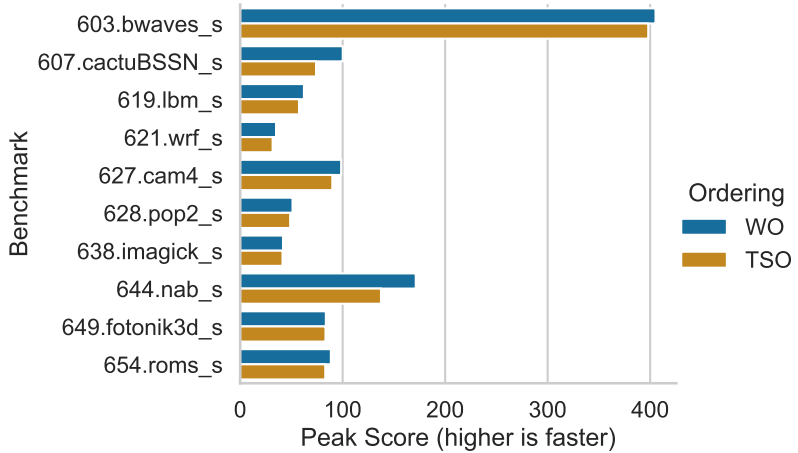


- ARMv8.3, Supports WO and TSO (by setting first bit of the **ACTLR_EL1**), 128 GiB

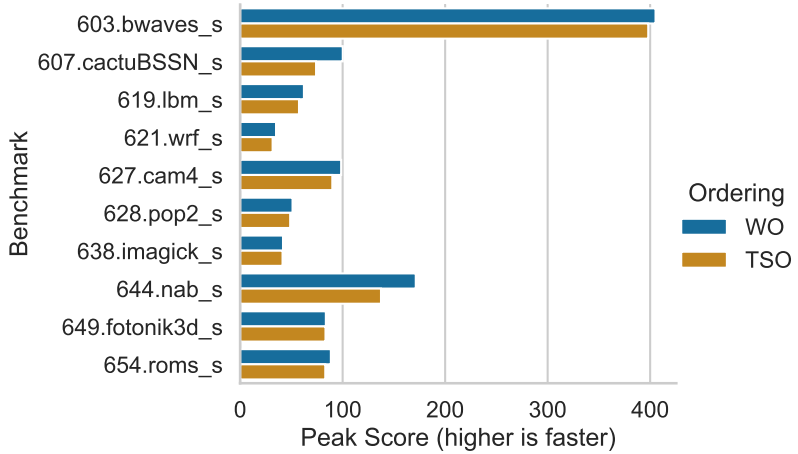
SPEC CPU 2017 Benchmarks

- Standardized benchmarking suite with realistic workloads
  - CPU- and memory-heavy applications (3D simulation, physics, image manipulation)
- We focus on multicore benchmarks
- Approach: Execute the same benchmark binaries with and without TSO

SPEC CPU 2017 Benchmarks

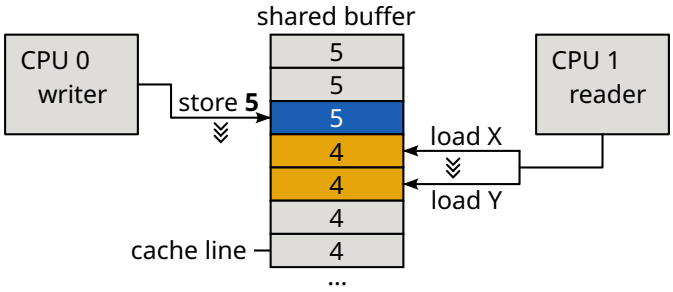


SPEC CPU 2017 Benchmarks



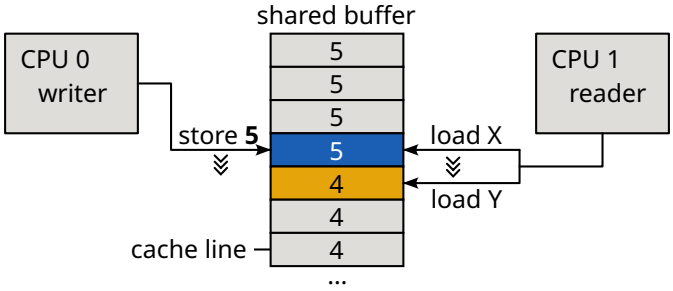
⇒ TSO is 8.94 % slower!

Store Micro-Benchmark

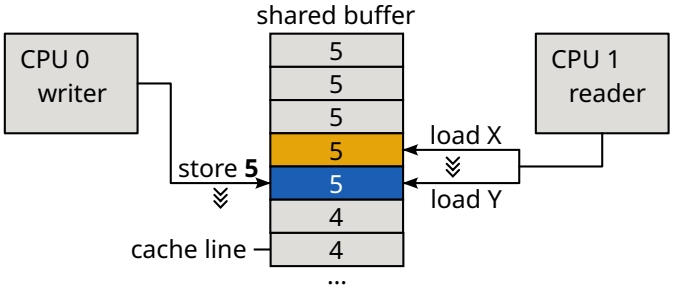


- Reader and writer iterate sequentially through the buffer
- Counting the number of completed iterations

Store Micro-Benchmark

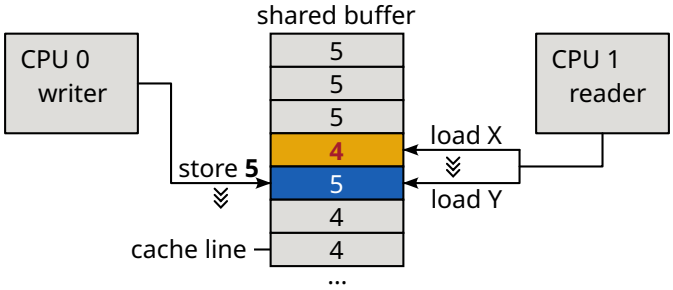


- Reader and writer iterate sequentially through the buffer
- Counting the number of completed iterations



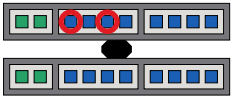
- Reader and writer iterate sequentially through the buffer
- Counting the number of completed iterations

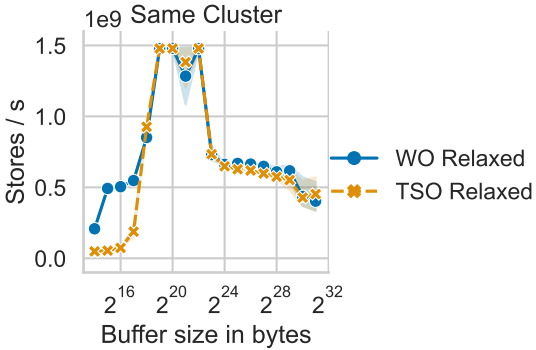
Store Micro-Benchmark



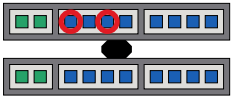
- Reader and writer iterate sequentially through the buffer
- Counting the number of completed iterations

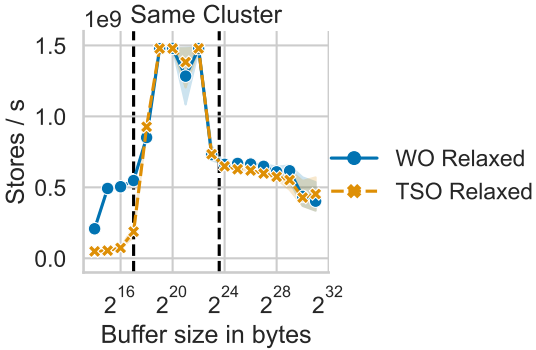
Store Micro-Benchmark



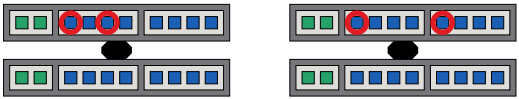


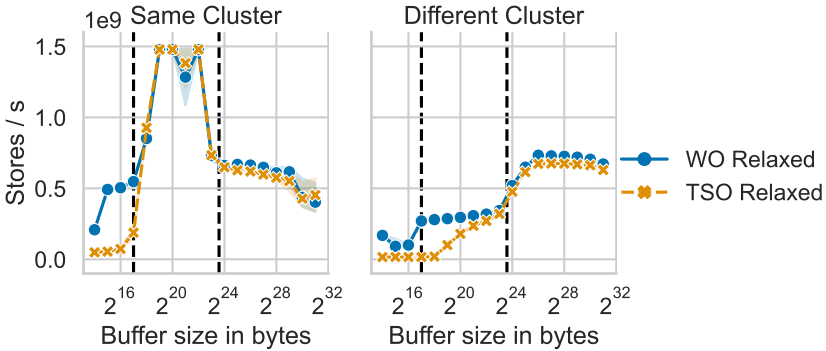
Store Micro-Benchmark

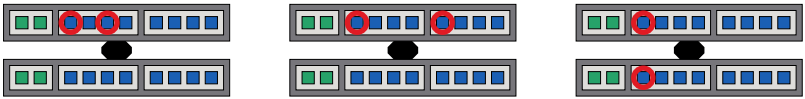


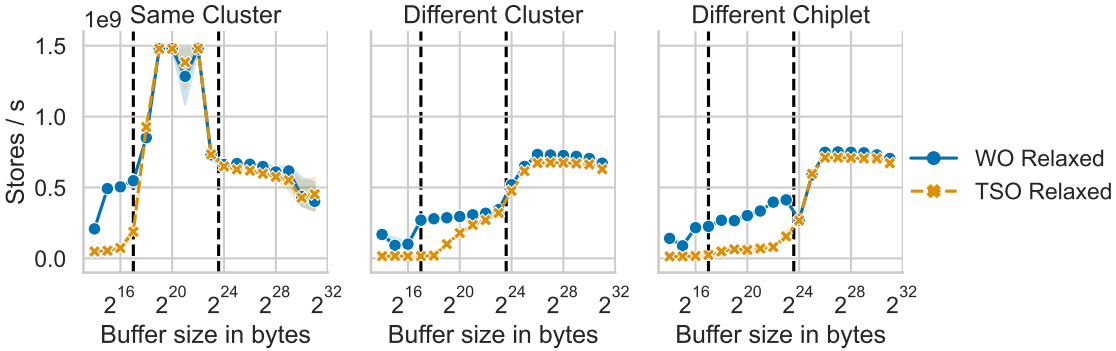


Store Micro-Benchmark

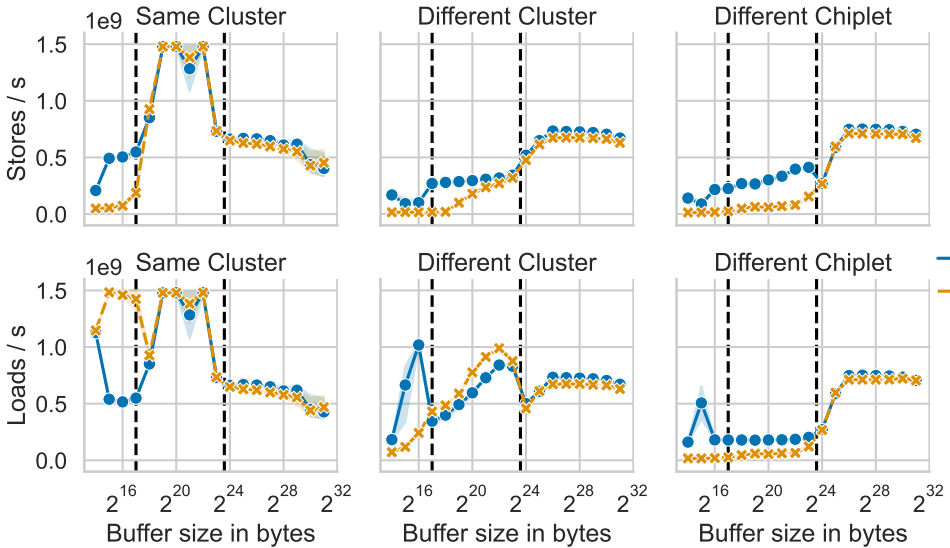




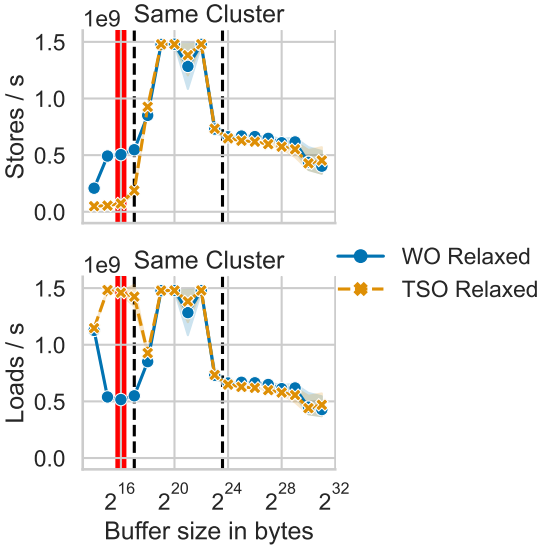


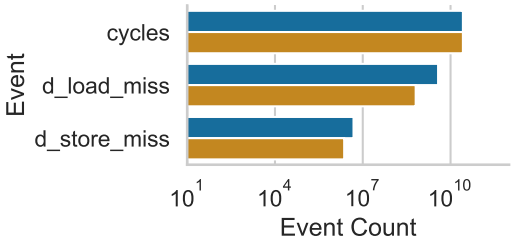

1e9
Same Cluster
Different Cluster
Different Chiplet
Stores / s
1.5
1.0
0.5
0.0
2^{16} 2^{20} 2^{24} 2^{28} 2^{32}
Buffer size in bytes
WO Relaxed
TSO Relaxed

Store Micro-Benchmark



Store Micro-Benchmark – Cache Misses



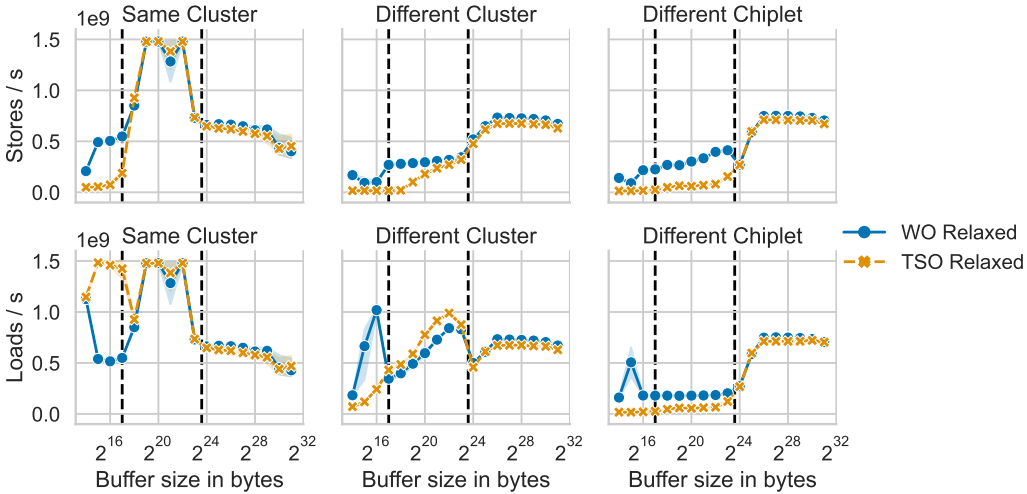


Weak stores are faster

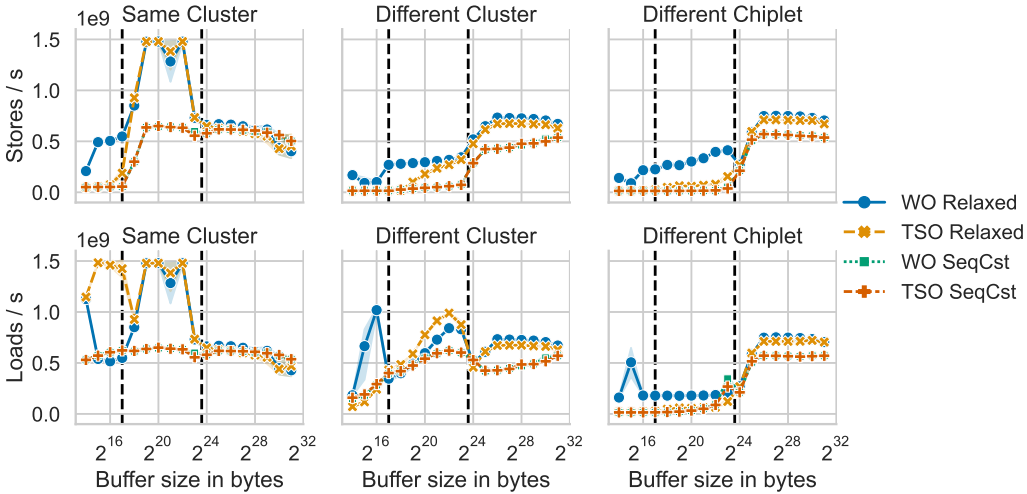
→ More cache invalidations

→ More load misses

Store Micro-Benchmark – Atomics



Store Micro-Benchmark – Atomics



- Sequentially consistent instructions (**STLR**, **LDAR**) are independend from TSO

Conclusion

- SPEC CPU 2017
  - TSO is 8.94 % slower
- Microbenchmarks
  - WO is faster for writes on small buffers
  - TSO is faster reads when stores are slow
- Sparse Information about the M1 Architecture

How to Investigate a Black Box?

- Microarchitecture is being reversed by the community (Johnson 2023)
  - Measuring Load/Store buffer sizes
- Asahi Linux developers have documented most registers (*Asahi Linux Wiki* 2023)
- Cache hierarchy not documented
  - We have reproduced L1/L2 cache sizes
  - Cacheline size is reported as 128 B but is actually 64 B

References I

- *Asahi Linux Wiki* (2023). `https://github.com/AsahiLinux/docs/wiki` – accessed 2023-03-23. (Visited on 03/23/2023).
- Gharachorloo, Kourosh et al. (1990). "Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors". In: *SIGARCH Comput. Archit. News* 18.2SI, pp. 15–26.
- Higham, Lisa et al. (1997). "Defining and comparing memory consistency models". In.
- Johnson, Dougall (2023). *Apple M1 Microarchitecture Research*. `https://dougallj.github.io/applecpu/firestorm.html` – accessed 2023-03-23. (Visited on 03/23/2023).
- Pulte, Christopher et al. (2017). "Simplifying ARM Concurrency: Multicopy-Atomic Axiomatic and Operational Models for ARMv8". In: *Proc. ACM Program. Lang.* 2.POPL.