Analysis of Software-Maintained Cache Coherency in ARMv8-A for Cross-Architectural DSM Systems: The Quintessential

Zhengyi Chen

April 25, 2024

Why study this, specifically?

- Amir Noohi (Prof. Barbalace's PhD student) currently works on in-kernel distributed shared memory system implemented over RDMA.
- We agree that cache coherency implementation and overhead is an important consideration for such a system to be cross-architectural (e.g., x86 vs. ARM64).
- Unlike x86, ARM64 (as well as e.g. RISC-V) does not guarantee hardware cache coherence:
  - ARM's "SystemReady" program requires such CPU/SoCs to be cache coherent at hardware level.
  - However, to my knowledge this is not the case across all ARM SoCs, especially as ARM64 PCs are becoming more prevalent.

Contributions

In this paper, I:

- Identified and exposed the in-Linux-kernel cache coherence maintenance implementation for ARM64 architecture (as used by e.g. RDMA drivers);
- Wrote a kernel module to perform performance tests on exposed mechanism, over both virtualized and server setups.

Cache Coherence in ARM64 & Linux Kernel

- ARMv8-A/R defines *Point-of-Coherence* (*PoC*) as the point at which all observers of memory will observe the same copy of a memory location.
- In Linux kernel (v6.7.0), this was in turn implemented as assembly macro `dcache_[clean|inval]_poc`, inside *arch/arm64/mm/cache.S*.
  - This was then called by Linux kernel's DMA API, e.g. `dma_sync_single_for_cpu`.
- For testing purposes, expose the assembly macro inside a C function symbol wrapper for dynamic ftrace support.

Experiment Setup: Kernel Module

- A simple kernel module for shared memory is written to test the latency of the exposed assembly macro via ftrace/bpf.
- A character device is exposed to userspace with `mmap` support.
  - Allocation size on driver end can be adjusted dynamically for testing cache coherence latency over variable-sized contiguous allocation.
  - Userspace programs can then adjust size of `mmap` for testing cache coherence latency over non-contiguous allocations.
- On `.close` (e.g. on `munmap` by userspace), allocations are flushed for *PoC* coherency...
  - As is the case for DMA memory, described prior.

Experiment Setup: Testbench

- Tests conducted mostly on QEMU virt-8.0 platform on x86 host.
- Some tests conducted on *Ampere Altra* server system...
  - However, *Ampere Altra* is *SystemReady*-certified – i.e., supports hardware level cache coherence.
  - Latencies collected on this system hence may be non-representative of real performance.
- Ideally wider range of test setups should be explored beyond the contributions of this paper.

Results: Summary

- Constant allocation size, variable `mmap` size: no significant difference in per-contiguous-memory-area cache coherence latency.
- Variable allocation size:
  - Latency **does not** grow linearly with increase in contiguous allocation size.
  - In general, latency remains within the same order-of-magnitude up to 2^6 contiguous pages.
  - For larger contiguous pages, latency due to cache coherence may be amortized by less allocations and page fault handlings required (implementation-specific?).
- In general, a hypothetical DSM system should prefer using larger contiguous allocations (which seems logical).