# Systematic Reverse Engineering of Cache Coherence Protocols Through Synthetic $\mu$-benchmarks

*CS246 Project Proposal*

Michael Buch, Yeongil Ko

**Abstract**

The world is multi-core. Reasoning about multi-core systems is hard. Multiprocessors rely on cache-coherence protocols (CCP) to coordinate private and shared memory. While these protocols are simple to reason about through finite-state machines, their implementation often is critical to the performance capabilities of a chip since they dictate the data movement of the entire system. Unfortunately, the details of how a processor maintains cache coherence is often hidden in incomprehensible prose or completely left out from its manuals. To aid the wider community of architecture researchers and system developers, our work describes a methodology to reverse engineer CCPs through micro-benchmarking and profiling multicore architectures; we do so on the following three platforms: (1) Intel Xeon Platinum (2) Intel Core i5 (3) ARM Cortex-A72

## 1 Introduction

Generally chip manufacturers reveal very little about the micro-architectural details in newer generation processors and mostly describe micro-architecture in the abstract. For example the Intel Architecture Optimization Manual [1] describes coherence only in the context of their TSX extensions or for their *Knight's Landing* architecture and other micro-architectures completely lack discussion of a CCP. The only obvious mention of managing coherence is that "a directory is used for cache coherence" without elaboration of implementation details. Understanding cache coherence is critical for architecture researchers and application and system developers in our increasingly multi-core world. Management of caches is management of data-movement and locality which are pillar-stones of modern-day system optimization. More formal studies, such as Peter Sewell's work [2, 3], have repeatedly shown the inability of architecture manuals to correctly describe coherence and memory system guarantees (including ARM, AMD and x86 architectures).

Cache-coherence is not the only micro-architectural feature that is mystified within architecture manuals. Another example is the branch-prediction unit. Recent reverse engineering efforts due to Google's Project Zero[1] and Maurice et al. [4] uncovered serious vulnerabilities in the processor branch predictor on Intel's Haswell, ARM Cortex, and AMD FX architectures and Intel's memory coherence respectively.

Instead of relying on manufacturers to provide cache-coherence details in the future, we propose a systematic approach to reverse engineering under-specified cache-coherence of modern processors: use micro-benchmarks to stress and map out a coherence protocol given specifications of the chip's memory hierarchy. Extensive work has been done on this in the security community for individual architectures. We hope to collate the approaches into a survey, ap-

[1]https://googleprojectzero.blogspot.com/2018/01/reading-privileged-memory-with-side.html

ply these techniques to a number of processors and think about a generalized methodology that would work across processor types.

The **aim of our project** is to reverse engineer and describe a close approximation of Intel Xeon Platinum's cache-coherence protocol. We will do so by writing micro-microbenchmarks and measuring performance counter events of the interconnects and memory subsystems. From this we will map out a finite state machine that approximates the cache-coherence protocol. We validate our benchmarking methodology by reverse-engineering an ARM processor for which a cache-coherence protocol specification is available.

## 2 Deliverables

- Gain insight into inner workings of cache coherence protocols (CCP) of ARM and Intel
- Create a set of micro-benchmarks that stress CCPs
- Evaluation of how viable our methodology is when applied to a wider range of platforms
- A proposal of how CCP micro-benchmarks could be generalized

## 3 Methodology

We follow the methodology described by Molka et al. [5, 6] in their investigation of cache coherence on the Intel Nehalem and Haswell architectures. The details we are after are inter-core communication latencies, inter-core communication bandwidth and, if possible, the actual CCP transition diagram.

Our methodology will consist of:

- Collect all reference data about memory hierarchies of Intel's Xeon and Core processors and ARM's Cortex processors. This will guide the structure of the workload in our benchmarks and what events we will be measuring. From preliminary research this looks to be a simple MESI or MESIF protocol [5, 6].
- Develop following types of benchmarks (measurements are done through performance counters/perf):
  1. Bandwidth Test: measure the total amount of data read/written across N number of threads to shared and private memory. Additionally, use different types of locks for synchronization to stress coherence. E.g., ones with stampede, single-reader, multiple-reader, with local or global locks, etc.
  2. Latency Test: measure the read/write time when accessing data on thread X from another thread Y
  3. State transition tests: if time allows, create benchmarks which will allow us to map out a state transition diagram. The benchmark should put caches into a known state (as done in [5]) and send data explicitly between core caches or force cache evicitions. Then perform the same set of data reads or writes and confirm, through latency and bus traffic measurements, whether/how the states of the caches changed

- We run our benchmarks on following three platforms:
  1. Intel Xeon Platinum[2]
  2. Intel Core i5[7]
  3. ARM Cortex-A72[8]
- We use the ARM Cortex-A72 as a test oracle for our methodology since details of its coherence protocol are readily available
- Evaluate whether our methodology could be mechanized or be applied to a wider range of coherence protocols such as the Dragon Protocol [9], GPU CCPs [10] or even data movement/cache coherence between accelerators on an SoC [11]

## 4 Workplan

- **November 5th - November 8th**: Set up infrastructure for developing and running benchmarks. This includes ensuring the necessary tooling such as *perf* and Pin is installed on RPI 4, our Linux machine and the condor server. Make sure performance counters are enabled on these machines. Also prepare and understand gem5's coherence models for the ARM Cortex series.
- **November 9th - November 16th**: Collect all publicly available information on coherence management in Intel Xeon, Intel Core i5 and ARM Cortex-A72 architectures. Make sure the benchmark methodology as described in Molka et al. [5] works on our infrastructure. This involves writing small benchmarks that successfully initializes caches on individual cores into the states we want. The states we choose will be dictated by the information we find about the cache hierarchy.

  Finalize the micro-architectural features we intend to measure in this period too.
- **November 17th - November 22nd**: Develop a micro-benchmark suite that stresses the cache hierarchy for the ARM and Intel Xeon processor. This work can be done in parallel between the two team members. Evaluate the performance of the memory subsystems and validate the results against publicly available data.
- **November 23rd - November 28th**: Infer a CCP and its micro-architectural details from the results and validate our approximation against official manuals (for ARM). Develop a plan to generate such micro-benchmarks automatically or make them more generalizable to work for a wider range of platforms.

  *This workplan accounts for 10 days of contingency*

[2]https://www.hashrates.com/cpus/intel(r)-xeon(r)-platinum-8275cl-48-3.00/

## References

[1] P. Guide, "Intel® 64 and IA-32 architectures software developer's manual," *Volume 3B: System programming Guide, Part*, vol. 2, 2011.

[2] P. Sewell, S. Sarkar, S. Owens, F. Z. Nardelli, and M. O. Myreen, "x86-TSO: a rigorous and usable programmer's model for x86 multiprocessors," *Communications of the ACM*, vol. 53, no. 7, pp. 89–97, 2010.

[3] H. J. Boehm, U. Goltz, H. Hermanns, and P. Sewell, "Multi-core memory models and concurrency theory (dagstuhl seminar 11011)," in *Dagstuhl Reports*, vol. 1, no. 1. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2011.

[4] C. Maurice, N. Le Scouarnec, C. Neumann, O. Heen, and A. Francillon, "Reverse engineering Intel last-level cache complex addressing using performance counters," in *International Symposium on Recent Advances in Intrusion Detection.* Springer, 2015, pp. 48–65.

[5] D. Molka, D. Hackenberg, R. Schone, and M. S. Muller, "Memory performance and cache coherency effects on an Intel Nehalem multiprocessor system," in *2009 18th International Conference on Parallel Architectures and Compilation Techniques.* IEEE, 2009, pp. 261–270.

[6] D. Molka, D. Hackenberg, R. Schöne, and W. E. Nagel, "Cache coherence protocol and memory performance of the Intel Haswell-EP architecture," in *2015 44th International Conference on Parallel Processing.* IEEE, 2015, pp. 739–748.

[7] O. Lempel, "2nd generation Intel® core processor family: Intel® core i7, i5 and i3," in *2011 IEEE Hot Chips 23 Symposium (HCS).* IEEE, 2011, pp. 1–48.

[8] A. ARM, "Cortex®-A72 MPCore processor technical reference manual (revision r0p2)."

[9] R. R. Atkinson and E. M. McCreight, "The dragon processor," in *ACM SIGARCH Computer Architecture News*, vol. 15, no. 5. IEEE Computer Society Press, 1987, pp. 65–69.

[10] I. Singh, A. Shriraman, W. W. Fung, M. O'Connor, and T. M. Aamodt, "Cache coherence for gpu architectures," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA).* IEEE, 2013, pp. 578–590.

[11] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, A. Knies, P. Ranganathan *et al.*, "Google workloads for consumer devices: Mitigating data movement bottlenecks," in *ACM SIGPLAN Notices*, vol. 53, no. 2. ACM, 2018, pp. 316–331.