15-418/618 Spring 2025: Project Proposal
Multi-processor Cache Simulator

Oluwaseyi Bello (obello) and Darshil Kaneria (dkaneria)

March 25, 2025

URL

Project web page: `https://darshilkaneria.site/csim/`

Summary

We plan to develop a multi-processor cache simulator that can model the behavior of caches in multi-core systems. Our implementation will follow the snooping-based design for cache coherence. After that, we will extend the implementation to also support a directory-based approach. The simulator will help analyze performance metrics such as cache hit/miss rates, interconnect traffic, bus utilization, and protocol overheads in various workloads. This project aims to provide insights into how different coherence protocols affect system performance in multiprocessor environments.

Background

In multi-processor systems, each processor typically has its own private cache, which creates the potential for multiple copies of the same memory block to exist simultaneously. When one processor modifies its copy, other copies become stale, leading to data inconsistency. Cache coherence protocols are one way to maintain consistency between caches. The two primary approaches are:

- **Snooping-based protocols**: Each cache controller monitors (or ”snoops”) the bus for transactions that might affect its cached data. Some common snooping protocols include:
  - **MSI**: The basic protocol with Modified, Shared, and Invalid states
  - **MESI**: Extends MSI with an Exclusive state for unshared clean blocks
  - **MOESI**: Further extends MESI with an Owned state for sharing modified data
  - **MESIF**: Extends MESI with a Forward state to optimize sharing
- **Directory-based protocols**: A centralized directory (in a NUMA system) tracks the sharing status of each memory block, reducing bus traffic by directing coherence messages only to relevant processors.

The Challenge

Workload Characteristics

There are a few patterns that we would like to evaluate to make sure our implementation is able to handle a variety of scenarios. For our workloads, we plan on having at least one trace from each of the following categories:

- We would like to test our simulator in a few obvious scenarios such as high-sharing and low-sharing workloads as well as a mixed one with no sharing

- We would also like to introduce a degree of randomness in some workload traces
- Testing a strided pattern (which is more common than other scenarios) will also be considered.
- Testing an Open-MP/MPI-like workload where a ”producer” writes and the ”consumers” read is also important
- We will perform final evaluation on traces generated from real-life workloads.

Resources

We plan to implement the cache simulator in C++. This is done to make sure the simulator resembles a low-level system as well as for the ease of programmability given the time constraints. We do not see ourselves requiring special machines to test our simulator on.

Goals and Deliverables

Plan to Achieve (75% grade target)

- Implement a working multi-processor cache simulator that correctly models standard MSI/MESI coherence protocol
- Parallelize the simulation to utilize at least 4-8 cores effectively
- Do a comprehensive analysis of simulator performance and accuracy

Hope to Achieve (100% grade target)

- Implement and evaluate multiple coherence protocols (MOESI, MESIF)
- Implement a directory scheme.
- Compare and contrast performance differences between snooping based and directory based caching schemes.

Fallback Goals (if progress is slower than expected)

- Focus on correctly simulating a simplified coherence protocol
- Provide detailed analysis of bottlenecks and challenges even if performance goals aren’t fully met

For the poster session, we plan to show performance graphs comparing our parallel implementation against baselines cache simulator, along with visualizations of cache behavior under different workloads.

Platform Choice

We’ve chosen to implement our simulator on multi-core CPU systems using C++ because we need a low-level systems language that gives us useful abstractions so that we can focus more on the analytical and experimentation part and less on minute implementation details.

Schedule

- **Week 1 (March 25 - March 31):**
  - Set up project repository and documentation
  - Implement basic sequential cache simulator framework
  - Design data structures for cache hierarchy representation
- **Week 2 (April 1 - April 7):**
  - Implement MSI/MESI coherence protocol in sequential simulator
  - Create test cases and validation framework
- **Week 3 (April 8 - April 14):**
  - Implement MESIF/MOESI
  - Implement a Directory based scheme
  - Prepare milestone report
- **Week 4 (April 15 - April 21):**
  - Develop benchmark suite for performance evaluation
  - Conduct initial performance measurements
- **Week 5 (April 22 - April 28):**
  - Perform final optimizations based on performance analysis
  - Create visualizations for result presentation
  - Prepare final report and poster