15-418 Project Proposal: Coherence Watchdog

Emily Szabo & Sophia Qingyang Cao | Mar 26, 2025

Website URL

https://github.com/emilyszabo27/15418_project

Summary

Our project aims to implement a snooping-based cache coherency monitor starting from the cache coherency lab starter code from 15-346. This includes implementing MSI, MESI, MESIF, and MOESI protocols.

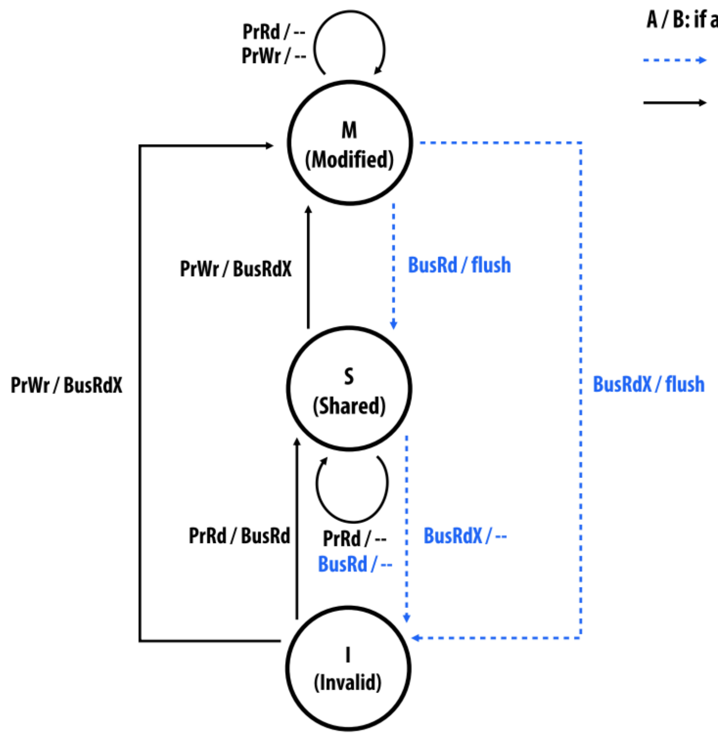
Background

In uniprocessor systems, caches can simply load/store and evict data without having to notify other parts of the system about their actions; however, when you move into parallel systems, this becomes a lot more challenging because all of the processors in the system have to communicate with each other regarding what data they’re writing into or reading from to maintain memory consistency across all of their caches.

Different protocols have been implemented for caches across multiprocessor systems to maintain memory consistency including snooping-based and directory-based protocols. Our project will focus on implementing several snooping-based protocols to gather stats about a multiprocessor system’s caches. Snooping-based protocols maintain memory consistency by storing metadata about each cache line: each line can be in a particular state, and depending on networking traffic from the cache controller or the interconnect between caches, cache lines can transition states. The following diagrams (credit to the

15-418 S25 lecture slides) are state diagrams for the MSI and MESI protocols.

MSI state transition diagram *



A / B: if action A is observed by cache controller, action B is taken

- - - → Remote processor (coherence) initiated transaction

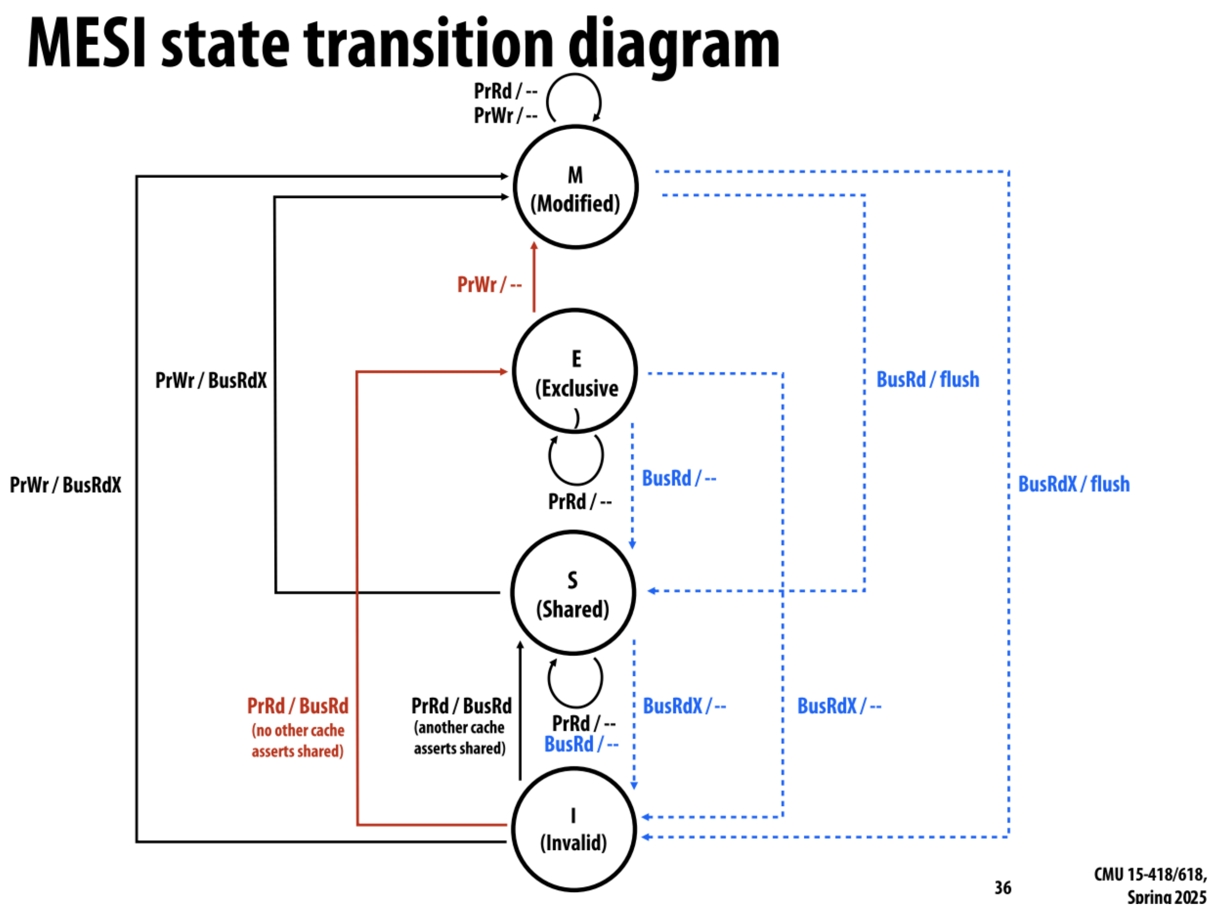
⟶ Local processor initiated transaction

flush = flush dirty line to memory

* Remember, all caches are carrying out this logic independently to maintain coherence

29

CMU 15-418/618, Spring 2025



We also plan to implement extended versions of these frameworks, MESIF and MOESI, whose diagrams are derived from the MESI diagram, each protocol extending the MESI protocol by a single state.

The Challenge

This is an interesting problem to implement because of the communication and atomicity complexities: there are many opportunities for incorrect outcomes to occur relating to ordering cache operations, especially writes.

It is difficult to implement a protocol for a multiprocessor system that abstracts a uniprocessor system because in reality there is both global memory that all caches can see and local per-cache storage that needs to be utilized intelligently to correctly implement this abstraction. We are really excited to dive deep into these synchronization issues to learn more about low-level system cache management!

Since this is a cache coherency implementation, the workload that we will be working on is generated traces for simulating a user's memory accesses, so the memory accesses for our implementation will utilize each of our simulated cache's local memory and global memory, as discussed earlier, as a complex synchronization problem.

This problem is constrained by the definition of coherency given in the lecture: (1) a read by processor P to address X that follows a write by P to address X should return the value of the write to P, (2) a read by processor P1 to address X that follows a write by processor P2 to X returns the written value if the memory accesses are sufficiently separated in time, and (3) writes to the same address are serialized, and two writes to address X should be observed as having the same order by all processors.

Resources

We will use the starter code specified here as the basis of our project, and we will use the shark machines as our computational mode for implementation. Since there is already a driver and a ref in the starter code, we do not need any more resources for this project.

Goals and Deliverables

We plan to achieve full implementation of MSI, MESI, MESIF, and MOESI protocols as well as designing further testing algorithms to produce traces that test specific tricky synchronization behavior to further show the robustness of our implementation.

Our stretch goals include gathering metrics to analyze the behavior as we scale up the processor count and running stress tests at higher processor counts.

We are hoping to learn a lot about the network activity across the cache interconnect as well as greatly improve our understanding and skill regarding the implementation of complex synchronization algorithms. In the end, we are looking forward to proudly presenting correct snooping-based cache coherency protocols for multiprocessor systems.

Platform Choice

We plan to code this project in C and use the shark machines because the starter code for this 15-346 project is built on C and utilizes the shark machines.

Schedule

WEEK 1 [Midterm II + Carnival week - lighter load]

- Ramp up to the code base (read through the project write-up and starter code to gather a thorough understanding of what is already given to us).

WEEK 2 [Toughest week - get this out of the way to move smoothly through the rest of the project schedule]

- Implement a correct MSI protocol.
- Complete the Project Milestone report to prepare for the meeting in WEEK 3.

WEEK 3 [Second toughest week - finish correct implementation strong, this *should* fall more smoothly into place after WEEK 2]

- Implement correct MESI, MESIF, and MOESI protocols.

WEEK 4 [Lighter week, before finals week to accommodate to general pre-finals week insanity]

- Design a test driver to exploit unique, tricky behavior that we want to show is robust in our implementation.
- Work on Final Project Report + Poster.

PRESENTATION WEEK [Lightest week, super short ½ week to wrap up project]

- Finalize project documentation (Final Project Report + Poster).

- Present work!