15-418 Final Project Checkpoint

Sam Flattery (sflatter) Brian Wei (bwei1)

December 1, 2020

Project Website: https://samflattery.github.io

1 Work Summary

We have a fully functional NUMA cache simulator using directory based cache coherence with the MESI cache protocol. We built this up incrementally, starting with a simple cache simulator for a single processor machine. We then added a directory and an interconnect that the directory and caches communicate through, which meant we could simulate a multiprocessor SMP machine, and added the MESI protocol to the cache blocks. We then further expanded the project by adding multiple directories and interconnects, with each NUMA node having its own directory and interconnect. The cache simulator also provides statistics about the trace it was ran on. As of now, it tracks hits, misses, evictions, invalidations, and dirty evictions for each cache, as well as interconnect events between caches and directories and between different NUMA nodes for each node.

We also expanded the sample `pin` program given on the Intel website which instruments the binary to record memory reads and writes, by recording the thread ID that made the read or write in order to simulate which CPU cache the memory will be sent to. We also added a probabilistic method of recording which NUMA node each memory access is on, so that our traces could have both NUMA and CPU information.

2 Updated Schedule

Week	Task	Completed
Week 1	• Meet with course instructors and present our ideas • Submit proposal • Study directory based cache coherence and come up with design ideas	✓
Week 2	• Get `pin` tool working and generate small sample trace files • Implement the cache simulators (i.e. tag bits, dirty bits, evictions, etc.)	✓
Week 3	• Get single directory working for the uniform memory access case • Directory should count communication events, implement MESI	✓
Week 4	• Expand system to have multiple distributed directories on NUMA architectures • Get `pin` to generate NUMA traces	✓
Week 5a	• Sam - Add new cache block types like MSI / MESIF / MOESI • Brian - Implement multi-leveled caches	
Week 5b	• Sam - Add in timing simulation for interconnect events, cache misses, etc. • Brian - Write additional parallel programs to trace and simulate	
Week 6a	• Both - Prepare final presentation, i.e. gather performance metrics and visualizations • Both - Prepare final writeup	
Week 6b	• Both - Prepare final presentation, i.e. gather performance metrics and visualizations • Both - Prepare final writeup	

3 Goals and Deliverables

We have completed all of the goals we planned to achieve (which mostly consisted of getting a fully working NUMA cache simulator that tracked stats of hits/misses/evictions) in our proposal and will be starting on our stretch goals after the checkpoint is submitted.

Our list of goals we plan to achieve before the poster session is as follows:

- Simulate simple timings of the cache events on top of counting metrics, such as fixing times for interconnect events, etc.
- Add MOESI / MESIF block types
- Implement multi-leveled caches
- Add more sample programs with different performance characteristics to compare the block types

Our deliverables consist of graphs and visualizations of the metrics provided by the simulator to compare the block types, so we plan to be able to produce these results after this week.

3.1 Deliverables

- Visualizations of the metrics provided by the simulator
- Analysis of sample programs under different cache conditions
- If timing implemented, differences between timings predicted by the simulator and actual program run times

4 Poster Session Plans

At the poster session, we plan to showcase a performance comparison between the various cache coherence protocols on several programs with difference characteristics. We will highlight aspects which are good or bad for cache performance, such as struct padding or false sharing.

On the poster itself, we plan on briefly explaining the various policies implemented, including state machine diagrams. Additionally, we will include some performance graphs for several programs, which we would explain in greater detail.

5 Preliminary Results

We wrote a sample program that spawns 8 threads and performs the following routine on an array of 1,000,000 elements

Our pintool creates a trace that looks like the following:

where the columns are thread ids (we assume thread IDs map uniquely to CPUs for simplicity), reads/writes, the memory address read/written to, and the NUMA node of that memory address respectively.

We run our program as follows: `./cachesim -t add.trace -s 6 -E 8 -b 6 -p 8 -n 2` which simulates the program on 8 processors split across 2 NUMA nodes, with 8 way associative caches with 64 byte cache blocks and 64 sets (i.e. a 32kB cache).

Our program outputs the following:

```
Running simulation with cache settings:
set size: 64
associativity: 8
line size: 64
```

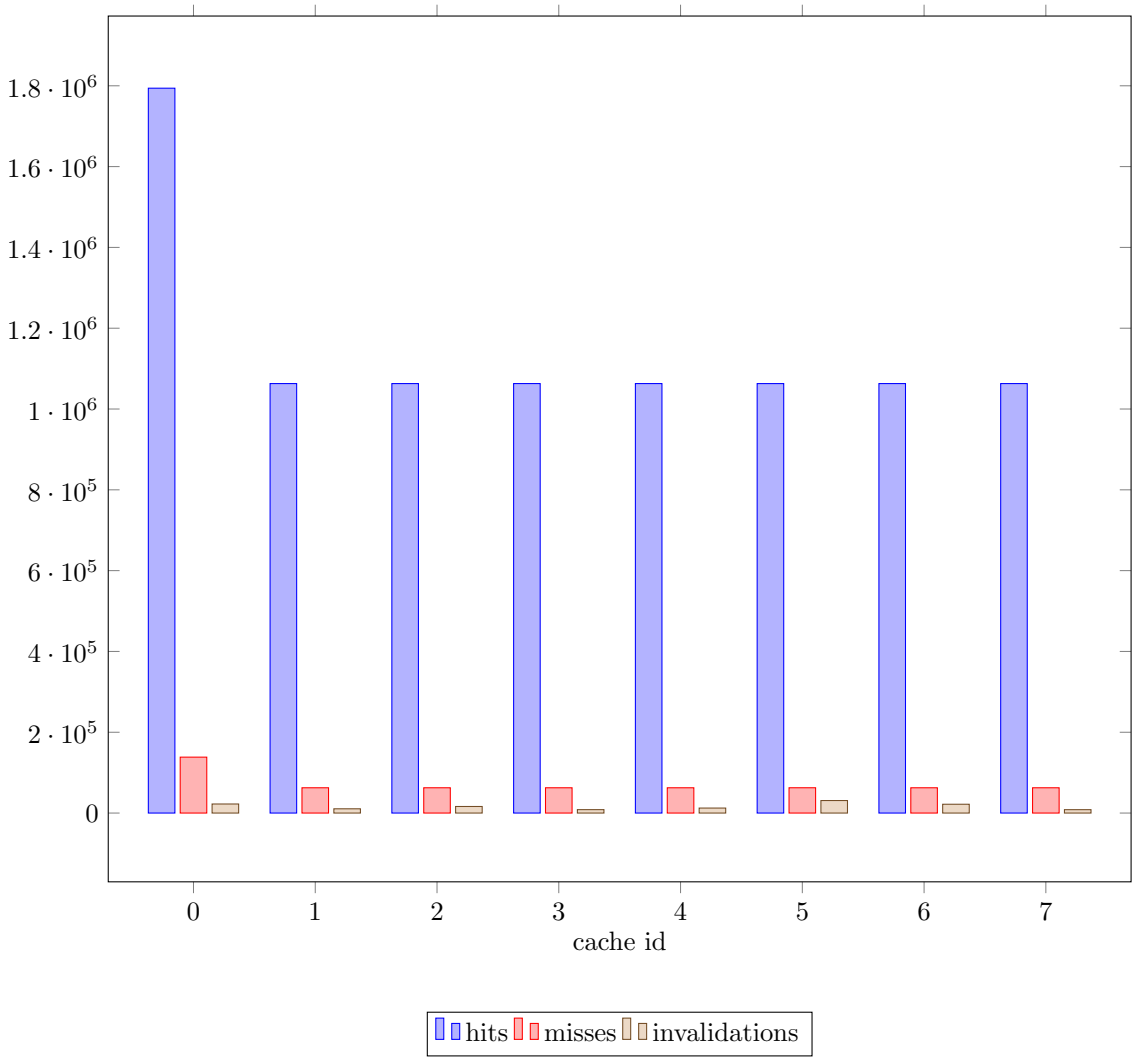
```
NUMA NODE: 0
-----------
*** Cache 0 ***
miss_count: 138395
hit_count: 1794063
eviction_count: 115593
invalidation_count: 22425
dirty_blocks_evicted: 115593

*** Cache 1 ***
...

*** Interconnect Events ***
Cache Events: 1008929
Directory Events: 441506
Global Interconnect Events: 409406
Total events: 1450435
```

Where cache events are messages sent from caches to the directory, directory events are messages sent from a directory to a cache, and global interconnect events are messages sent between NUMA nodes.

This results in the following distribution of cache hits/misses:



6 Concerns

We don't have any concerns beyond just the work we have to do to finish the project off. We believe we are on track to achieve our initially proposed deliverables as well as the additional goals we hoped to achieve.