Directory Based Cache Coherence Simulator

Sam Flattery (sflatter) Brian Wei (bwei1)

November 4, 2020

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

1 Summary

We are going to write a cache simulator for a multiprocessor machine with a NUMA architecture. The simulator will use distributed directory based cache coherence to maintain coherency between the caches. It will take a trace file of memory accesses and the processors making those accesses as input and provide statistics on the program's cache performance.

2 Background

NUMA (Non Uniform Memory Access) is a multiprocessing computer architecture in which memory access times depend on the proximity of the memory location to the processor requesting the access. It is often used in distributed machines, as it means that processors have faster access to their own local memory than they would have to a global memory shared across all processors, at the expense of occasional high latency access to memory located further away.

To ensure cache coherence under this architecture, a directory based coherence protocol is often used. The processor that owns a certain line of memory stores information about each processor that has a copy of the memory in its cache, or that has exclusive access to that memory. It can then send out point-to-point messages to the caches that hold the line whenever they must update their state. This provides a more scalable approach than snooping based coherence, where expensive broadcast messages would have to be sent to multiple processors.

3 Challenges

The main challenge is going to be accurately simulating the state of each cache directory – we anticipate some of the difficulties will be in generating precise metrics and accurate timing estimates from the simulator. We also find that

it may be difficult to account for different NUMA architectures including those with distributed directories.

4 Resources

We will be using Intel's `pin` tool in order to generate trace files. `pin` is a tool that can dynamically instrument a binary at runtime, which will allow us to insert code to write the trace files which will be triggered upon memory reads and writes.

We are considering using `QEMU`, a machine emulator, to emulate a NUMA machine so that we can generate trace files that also contain which NUMA node a memory address resides on.

5 Goals and Deliverables

5.1 Plan to Achieve

- Simulate the state transitions of a multiprocessor NUMA cache using directory based cache coherence
- Allow the user to customize the cache configuration, such as set sizes and associativity
- Allow the user to customize the NUMA configuration by specifying how many NUMA nodes, how many processors, etc.
- Provide metrics such as cache hits, misses, evictions, interconnect messages, etc., in order for the tool to present statistics on sharing

5.2 Hope to Achieve

- Simulate simple timings of the cache events as well as counting metrics, such as fixing times for interconnect events, etc.
- Allow the user to set parameters of the NUMA architecture, such as how far away NUMA nodes are
- Experiment with memory reducing techniques such as limited pointer schemes and sparse directories
- Compare different state transitions such as MOESI and MOSIF for different uses cases

5.3 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

6 Platform

We are not relying on any specialized hardware on which to run our simulator. We plan on implementing our simulator in C++ since `pin` uses C/C++ and in order to maximize performance of the simulator since the trace files will likely be very large (on the order of megabytes). It is also a language we are both familiar with.

7 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 • Add intervention / request forwarding	
Week 5	• Explore optimizations such as MOSIF / MOESI, sparse directories • Stretch goal - add in timing simulation for interconnect events, cache misses, etc.	
Week 6	• Prepare final presentation, i.e. gather performance metrics and visualizations • Prepare final writeup	