18-640: Foundations of Computer Architecture
Project 3
Due Sunday, November 15 11:59 p.m. EST

This project aims to expose real-life cache modeling to you. The project is divided into three parts. In the first part of the project, you will implement a cache replacement policy LFU (Least Frequently Used), in the second part, you will design a cache coherence protocol, specifically MSI (Modified-Shared-Invalid), and in the last part you will analyze coherence mechanisms i.e. directory-based and snooping-based.

Setup:

1. Copy `/afs/ece.cmu.edu/class/ece640/project/project3/lab3.patch` to your gem5 folder.
2. `patch -p1 < lab3.patch`

Please make sure you see a list of file changes in `stdout`. This patch applies minor changes to make it easier for you to get started with Project 3.

1 Part 1: Cache Replacement Policy

Caches have very limited size, therefore, in order to make room for the new entry on a cache miss, the cache may have to evict one of the existing entries. The heuristic that it uses to choose the entry to evict is called the replacement policy. The fundamental problem with any replacement policy is that it must predict which existing cache entry is least likely to be used in the future. The most common replacement policy is LRU (Least Recently Used) which is already present in gem5. Another common alternative is the LFU (Least Frequently Used) replacement policy, which you will implement.

Your Task:

Go through the codebase in `src/mem/cache/` and implement LFU (Least Frequently Used) cache replacement policy. Before starting to implement LFU, make sure you understand the LRU policy, specifically in `src/mem/cache/tags/{lru.cc, lru.hh}`. We have created the following dummy files for you to get started. It is exactly same as LRU and only the class name is modified. Please make modifications to only these files to implement LFU policy.

`src/mem/cache/tags/lfu.cc`, and,

`src/mem/cache/tags/lfu.hh`.

To analyze the replacement policies, you will be using PARSEC Benchmarks in FS mode, as you did in your previous project. Specifically, you need to report statistics for blackscholes benchmark, `runscript_1.rcS`, and x264 benchmark, `runscript_2.rcS`, running once with LRU policy and once with LFU policy. Please report an average of the results obtained from each of the two benchmarks in this format:

	system.cpu.cpi	system.cpu.icache.tags.replacements	system.cpu.dcache.tags.replacements
LRU			
LFU			

Note: You can switch cache replacement policy of the simulator by replacing *LRU* with *LFU* in the last line of `src/mem/cache/BaseCache.py`. Do not forget to re-build after every modification.

Command to run:

```
$ scons build/ALPHA/gem5.opt
```

```
$ build/ALPHA/gem5.opt configs/example/fs.py --script=<.rcS>script
  --cpu-type=detailed --caches
```

Report:

1. Between LRU and LFU, which is better? [2]
2. Analyze the results you obtained. [4]
3. Which of the two you think is easier to implement in hardware? Give reasons. [2]

2 Part 2: Cache coherence and SLICC

Cache coherence is the consistency of shared resource data that ends up being stored in multiple local caches. The protocol should define the behavior of reads and writes to the same memory location.

Types of states

Stable States of blocks that are not currently in the midst of a coherence transaction. The most common ones are:

M odified

O wned: The block is valid, owned, and potentially dirty, but not exclusive. The cache has a read-only copy of the block and must respond to requests for the block.

E xclusive: The block is valid, exclusive, and clean. The cache has a read-only copy of the block. No other caches have a valid copy of the block, and the copy of the block in the LLC/memory is up-to-date.

S hared

I nvalid

Transient Transient states arise when transitions between stable states are not atomic. For example, an individual message sends and receives are atomic, but fetching a block from the memory controller requires sending a GET message and receiving a DATA message, with an indeterminate gap in between. During such a ‘gap’, the cache block is said to be in a transient state.

gem5 - SLICC

SLICC is a domain specific language for specifying cache coherence protocols. The SLICC compiler generates C++ code for different controllers, which can work in tandem with other parts of Ruby. The SLICC compiler takes, as input, files that specify the controllers involved in the protocol. The `.slicc` file specifies the different files used by the particular protocol under consideration. For example, if trying to specify the MSI protocol using SLICC, then we may use `MSI.slicc` as the file that specifies all the files necessary for the protocol. The files necessary for specifying a protocol include the definitions of the state machines for different controllers, and of the network messages that are passed on between these controllers. All protocols are defined at `src/mem/protocol/`.

Required Reading

You must go through `http://www.m5sim.org/SLICC` in order to understand the different parts of the cache protocol specification.

Your Task

1. You need to understand the specifications in the following files and draw its state diagram.

   `src/mem/protocol/MI_example-cache.sm` and,

   `src/mem/protocol/MI_example-dir.sm` .

2. Now, assume you have been assigned to implement MSI protocol for gem5. Design its state diagram for cache controller and directory controller.
   Assume private, unified, L1-caches (i.e., both instruction fetches and memory requests access the same cache) with single directory controller.

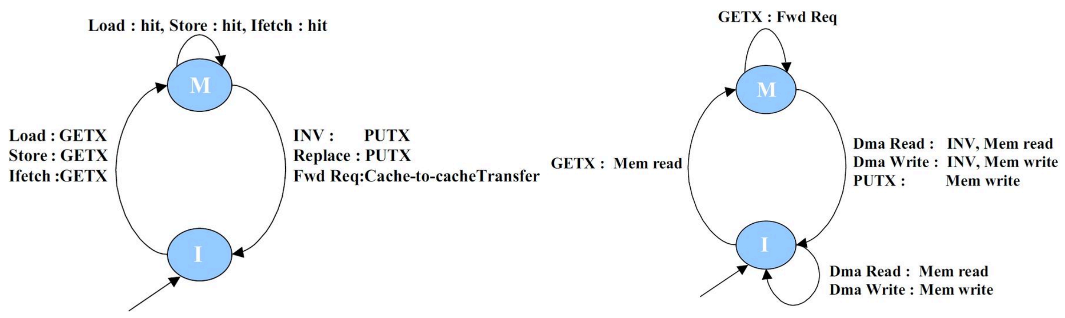

Figure 1: MI protocol description

Extra: We do not want you to code your MSI design, but consider doing so in your free time. You may consult your TAs for help regarding this.

Your state diagrams should be similar to Fig. 1, but should also include the transient states. If you submit hand-drawn figures, please scan them and include them in PDF format.

Report:

1. Why are transient states necessary? [2]
2. Why does a coherence protocol use stalls? [4]
3. What is deadlock and how can we avoid it? [2]

3 Part 3: Cache coherence - Protocol Analysis

The two main coherence mechanisms are:

Directory based The data being shared is placed in a common directory that maintains the coherence between caches. The directory acts as a filter through which the processor must ask permission to load an entry from the primary memory to its cache. When an entry is changed the directory either updates or invalidates the other caches with that entry.

Snooping based This is a process where the individual caches monitor address lines for accesses to memory locations that they have cached. It is called a write invalidate protocol when a write operation is observed to a location that a cache has a copy of and the cache controller invalidates its own copy of the snooped memory location. It is the default coherence mechanism.

Please go through the following links for more details about their modeling in gem5.

`http://www.gem5.org/General_Memory_System` and,

`http://www.m5sim.org/Interconnection_Network`.

Your Task

You now need to analyze *directory-based* protocols and *snooping-based* protocols for the jpeg-decode and quicksort benchmarks. You can use MOESI_CMP_directory as the directory-based protocol and for snooping-based, use the default one. Commands to run:

```
# To select the coherence protocol when building
scons PROTOCOL=MOESI_CMP_directory build/ALPHA/gem5.opt

# For snooping-based protocols
# Command to run jpeg-decode
build/X86/gem5.opt configs/example/se.py -c mibench/consumer/jpeg/jpeg-6a/djpeg\
-o "-dct int -ppm -outfile mibench/consumer/jpeg/output_small_decode.ppm\
mibench/consumer/jpeg/input_small.jpg" --cpu-type=detailed --caches -n 4 --num-l2caches=4

# Command to run quicksort
build/X86/gem5.opt configs/example/se.py -c mibench/automotive/qsort/qsort_small\
-o "mibench/automotive/qsort/input_small.dat"\
--output=mibench/automotive/qsort/output_small.txt--cpu-type=detailed\
--caches -n 4 --num-l2caches=4
```

To run directory-based protocols add `--ruby` to the end of the commands, so that it uses the `MOESI_CMP_directory` protocol.

Report:

1. Are the results intuitive? Give reasons. [3]
2. When is snooping-based desirable over directory-based, and vice-versa? [3]
3. What are the different topologies you can simulate in gem5? What are their advantages? Which one did you simulate i.e. the default topology? [7]

4 Grading

Your project will be graded as follows:

25%	Part 1: Cache Replacement Policy
(20 + 20)%	Part 2: Cache Coherence - SLICC (MI + MSI)
10%	Part 3: Cache Coherence - Protocol Analysis
25%	Report (9 Questions)

5 Hand-in Details

You will need to hand-in `lfu.cc`, `lfu.hh` and `report.pdf` to `/afs/ece/class/ece640/submission/<groupname>/project3/`.

If you have any doubt, please feel free to ask us. We are more than happy to help!