CS 211: Computer Architecture

Homework 6: Cache Simulator

Fall 2020

Overview

The goal of this assignment is to help you understand caches better by writing a cache simulator in C. The programs have to run on ilab machines. We are providing real program memory traces as input to your cache simulator. The format and structure of the memory traces are described below.

You can assume all your input files will be in proper format as described.

Memory access traces

The input to the cache simulator is a memory access trace, which we have generated by executing real programs. The trace contains memory addresses accessed during program execution. Your cache simulator will have to use these addresses to determine if the access is a hit or a miss, and the actions to perform in each case.

The memory trace file consists of multiple lines, each of which corresponds to a memory access performed by the program. Each line consists of multiple columns, which are space separated.

The first column reports the PC (program counter) when this particular memory access occurred, followed by a colon. Second column lists whether the memory access is a read (R) or a write (W) operation. And the last column reports the actual 48-bit memory address that has been accessed by the program.

In this assignment, you only need to consider the second and the third columns (i.e. you don't really need to know the PCs). The last line of the trace file will be the string "`#eof`". We have provided you a few input trace files (some of them are larger in size).

Here is a sample trace file:

```
0x804ae19: R 0x9cb3d40
0x804ae19: W 0x9cb3d40
0x804ae1c: R 0x9cb3d44
0x804ae1c: W 0x9cb3d44
0x804ae10: R 0xbf8ef498
#eof
```

Cache simulator

You will implement a cache simulator to evaluate different congurations of caches against different trace files.

- The cache has only one level, i.e., an L1 cache.
- The input parameters for cache size and block size are specfied in bytes.
- The cache is write-through.
- When your program starts, there is nothing in the cache. So, all cache lines are empty (invalid).
- The memory size is 2^{48} bytes. Therefore, memory addresses are 48 bits (zero extend the addresses in the trace file if they're less than 48-bit in length).
- The number of bits for the tag, set index, and block offset are determined by the input parameters.
- In the case of a write miss, assume that the block is first read from memory (one memory read), and then followed by a memory write.
- You do not need to simulate the memory itself in this assignment, since the traces don't contain any information on data values transferred between the memory and the caches.

Cache simulator interface

Your program should support the following command-line arguments:

```
./first <cache size> <associativity> <replace policy> <block size> <trace file>
```

where:

- `<cache size>` is the total size of the cache in bytes. This number should be a power of 2.
- `<associativity>` is one of:
  - `direct` – simulate a direct mapped cache.
  - `assoc` – simulate a fully associative cache.
  - `assoc:n` – simulate an n-way associative cache. n will be a power of 2.
- `<replace policy>` is the replacement policy, either `lru` or `fifo`.
- `<block size>` is a power of 2 integer that specifies the size of the cache block in bytes.
- `<trace file>` is the name of the trace file.

All command-line arguments (except possibly the trace file name) will be all lowercase.
Your program should check if all the inputs are in valid format, if not print "`error`" and exit.

Cache replacement policy

The goal of the cache replacement policy is to decide which block has to be evicted in case there is no space in the set for an incoming cache block. It is always preferable – to achieve the best performance – to replace the block that will be referenced furthest in the future. There are different ways one can implement the cache replacement policy, and we'll consider two.

Least recently used (LRU)

Using this algorithm, you always evict the block accessed least recently in the set without any regard to how often or how many times it was accessed before. Let's say that your cache is empty initially and that each set has two lines. Now suppose that you access blocks A, B, A, C. To make room for C, you would evict B since it was accessed less recently than A.

First-in first-out (FIFO)

In first-in-first-out, you evict the block that was added to the set earliest. Suppose we have two lines and access A, B, A, C. To make room for C, you would evict A, since it was added earlier than B.

Output

Your program should print out the number of memory reads, memory writes, cache hits, and cache misses.

```
$ ./first 32 assoc:2 lru 4 trace1.txt
Memory reads: 336
Memory writes: 334
Cache hits: 664
Cache misses: 336
```

Note: Some of the trace files are quite large, so it might take a few minutes for the autograder to grade all the testcases.