CS5541 - Computer Systems
Cachelab

Jason Eric Johnson

02/20/2019

1 Introduction

This is an individual project. The purpose of this assignment is to better understand the operation of cache memory. You will write a cache simulator that will keep track of hits, misses, and evictions.

2 Downloading the assignment

> Students can download the assignment files from the course Elearning site. The code archive file will be called `wmucachelab.tar.gz`. This file contains the file `csim-ref` and a directory called `traces` that contains the trace files that you will use to test your cache simulator. The `csim-ref` executable will run on the CS login servers and should run on any Ubuntu Linux system.

Start by copying `wmucachelab.tar.gz` to a protected Linux directory in which you plan to do your work and extract the archive. This will create a directory called `wmucachelab` that contains the files mentioned above.

3 Reference Trace Files

The `traces` subdirectory of the handout directory contains a collection of *reference trace files* that you can use to evaluate the correctness of the cache simulator you write. The trace files are generated by a Linux program called `valgrind`. For example, typing

```
linux> valgrind --log-fd=1 --tool=lackey -v --trace-mem=yes ls -l
```

on the command line runs the executable program "`ls -l`", captures a trace of each of its memory accesses in the order they occur, and prints them on `stdout`.

`Valgrind` memory traces have the following form:

```
I 0400d7d4,8
 M 0421c7f0,4
 L 04f6b868,8
 S 7ff0005c8,8
```

Each line denotes one or two memory accesses. The format of each line is

```
[space]operation address,size
```

The *operation* field denotes the type of memory access: "I" denotes an instruction load, "L" a data load, "S" a data store, and "M" a data modify (i.e., a data load followed by a data store). There is never a space before each "I". There is always a space before each "M", "L", and "S". The *address* field specifies a 64-bit hexadecimal memory address. The *size* field specifies the number of bytes accessed by the operation.

4 Writing a Cache Simulator

You will write a cache simulator that takes a `valgrind` memory trace as input, simulates the hit/miss behavior of a cache memory on this trace, and outputs the total number of hits, misses, and evictions.

I have provided you with the binary executable of a *reference cache simulator*, called `csim-ref`, that simulates the behavior of a cache with arbitrary size and associativity on a `valgrind` trace file. It uses the LRU (least-recently used) replacement policy when choosing which cache line to evict.

The reference simulator takes the following command-line arguments:

```
Usage: ./csim-ref [-hv] -s <s> -E <E> -b <b> -t <tracefile>
```

- `-h`: Optional help flag that prints usage info
- `-v`: Optional verbose flag that displays trace info
- `-s <s>`: Number of set index bits ($S = 2^s$ is the number of sets)
- `-E <E>`: Associativity (number of lines per set)
- `-b <b>`: Number of block bits ($B = 2^b$ is the block size)
- `-t <tracefile>`: Name of the `valgrind` trace to replay

The command-line arguments are based on the notation (s, E, and b) from Chapter 6 of your textbook. For example:

```
linux> ./csim-ref -s 4 -E 1 -b 4 -t traces/yi.trace
hits:4 misses:5 evictions:3
```

The same example in verbose mode:

```
linux> ./csim-ref -v -s 4 -E 1 -b 4 -t traces/yi.trace
L 10,1 miss
M 20,1 miss hit
L 22,1 hit
S 18,1 hit
L 110,1 miss eviction
L 210,1 miss eviction
M 12,1 miss eviction hit
hits:4 misses:5 evictions:3
```

Your job is to write a cache simulator so that it takes the same command line arguments and produces the identical output as the reference simulator. Notice that this assignment starts from a ”blank page.” You’ll need to write it from scratch.

5 Programming Rules

- Include your name and email address in header comments in your code.
- Your simulator must compile/interpret/run without warnings in order to receive credit.
- Your simulator must work correctly for arbitrary s, E, and b.
- For this assignment, we are interested only in data cache performance, so your simulator should ignore all instruction cache accesses (lines starting with "I"). Recall that `valgrind` always puts "I" in the first column (with no preceding space), and "M", "L", and "S" in the second column (with a preceding space). This may help you parse the trace.
- You may NOT modify the trace files. I will be testing your simulator with traces of the same form as the reference traces I'm providing you, so your simulator MUST work with those traces as they are.
- To receive credit you must clearly indicate the total number of hits, misses, and evictions at the end of each simulation.
- For this assignment, you should assume that memory accesses are aligned properly, such that a single memory access never crosses block boundaries. By making this assumption, you can ignore the request sizes in the `valgrind` traces.

6 Evaluation

This section describes how your work will be evaluated. The full score for this assignment is 90 points:

- Program Correctness: 81 Points
- Style: 9 Points

6.1 Evaluation for Correctness

I will run your cache simulator using different cache parameters and traces. There will be eight test cases, each worth 3 points, except for the last case, which is worth 6 points. The total number of points reported will be multiplied by three to weight the correctness score for the assignment to 81 points:

You can use the reference simulator `csim-ref` to test your simulator. Your simulator should return the same number of hits, misses, and evictions as the reference simulator. During debugging, use the `-v` option for a detailed record of each hit and miss.

Here are some examples of test runs using the reference simulator you will be given. You should be able to run your simulator with the same parameters as you see below (and any other set of parameters) and get the same results.

```
linux> ./csim-ref -s 1 -E 1 -b 1 -t traces/yi2.trace
linux> ./csim-ref -s 4 -E 2 -b 4 -t traces/yi.trace
linux> ./csim-ref -s 2 -E 1 -b 4 -t traces/dave.trace
linux> ./csim-ref -s 2 -E 1 -b 3 -t traces/trans.trace
linux> ./csim-ref -s 2 -E 2 -b 3 -t traces/trans.trace
linux> ./csim-ref -s 2 -E 4 -b 3 -t traces/trans.trace
linux> ./csim-ref -s 5 -E 1 -b 5 -t traces/trans.trace
linux> ./csim-ref -s 5 -E 1 -b 5 -t traces/long.trace
```

For each test case, outputting the correct number of cache hits, misses and evictions will give you full credit for that test case. Each of your reported number of hits, misses and evictions is worth 1/3 of the credit for that test case. That is, if a particular test case is worth 3 points, and your simulator outputs the correct number of hits and misses, but reports the wrong number of evictions, then you will earn 2 points.

6.2 Evaluation for Style

There are 9 points for coding style. These will be assigned manually by me. Criteria for style points will include proper indentation, consistent coding style, descriptive variable naming, etc. There are quite a few resources out there for coding style standards. I'd recommend finding one you like and following it. This is good practice for programming in general, and if you find one and specify it in your README.TXT (with a link so I can go find it) I can evaluate your code against those guidelines, which will help get you full style points for the assignment.

7 Working on the Assignment

Here are some hints and suggestions for working on the assignment:

- You may use C, C++, Java, Python, or Ruby for this assignment. If you don't like any of those and would like to use something else please clear your choice of language with me before you start. If you submit the assignment in a language not listed here without getting approval your submission will not be graded.
- Do NOT use any code from the internet or any other sources. Write your own, original code. If you look for examples of HOW to do something and then write your program once you understand it, then that's fine. Be sure, however, to include a reference to any resources you look at, online or elsewhere, in your README.TXT file. For those kinds of references indicate what code you looked at and what you wrote in your assignment after you got an idea of how it worked. References without explanation will not be considered valid and could lead to loss of points or, in the worst case, amount to academic dishonesty, so be complete in your explanation of how you used anything you looked at to increase your understanding.
- Do your initial debugging on the small traces, such as `traces/dave.trace`.
- The reference simulator takes an optional `-v` argument that enables verbose output, displaying the hits, misses, and evictions that occur as a result of each memory access. You are not required to implement this feature in your simulator, but I strongly recommend that you do so. It will help you debug by allowing you to directly compare the behavior of your simulator with the reference simulator on the reference trace files.
- Each data load (L) or store (S) operation can cause at most one cache miss. The data modify operation (M) is treated as a load followed by a store to the same address. Thus, an M operation can result in two cache hits, or a miss and a hit plus a possible eviction.

8 Handing in Your Work

Submit Cachelab using Elearning. There does not need to be any tracefile or .PDF submitted for this assignment. Just submit your simulator code to the Dropbox on Elearning. Please make sure you have a README.TXT that contains your name and email adress, any coding style guidelines you used, any reference material you used to help you write your simulator, and instructions indicating how I should go about running your code. Indicate what (if any) IDE you used, how I should import your code into it, what language you used, what version, etc.