CS211 Fall 2017

Prof. Santosh Nagarakatte
Programming Assignment 4: Simulate Caches
Due: 5:00 PM, Sunday December 3

Overview

The goal of this assignment is to provide you a better understanding of caches. You are required to write a cache simulator using the C programming language. The programs have to run on iLab machines and should be tested with the autograder.

We are providing real program memory traces as input to your cache simulator. The format and structure of the memory traces are described below.

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, miss, and the actions to perform.

The memory trace file consists of multiple lines. Each line of the trace file corresponds to a memory accesses 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

Here is a sample trace file:

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

Cache Simulator (100 points)

You will implement a cache simulator to evaluate different configurations of caches. It should be able to run with different traces files. The followings are the requirements for your cache simulator:

1. Simulate only **one level** cache: L1
2. The cache size, associativity, and block size are input parameters. Cache size and block size are specified in bytes.

3. Replacement algorithm: First In First Out (FIFO). When a block needs to be replaced, the cache evicts the block that was accessed first. It does not take into account whether the block is frequently or recently accessed..

4. It's a write through cache.

Running your Cache Simulator

You have to name your cache simulator first. Your program should support the following usage interface:

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

where:

A) < cachesize > is the total size of the cache in **bytes**. This number should be a power of 2.
B) < 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.

C) <cache policy> Here is valid cache policy is fifo.

D) < blocksize > is a power of 2 integer that specifies the size of the cache block in bytes.

E) < tracefile > is the name of the trace file.

Cache Prefetcher:

Prefetching is a common technique to increase the spatial locality of the caches beyond the cache line. The idea of prefetching is to bring the data into the cache before it is needed (accessed). In a normal cache, you bring a block of data into the cache whenever you experience a cache-miss. Now, we want you to explore a different type of cache that prefetches not only brings the block corresponding to the access but also prefetches **one adjacent block,** which will result in one extra memory read. An adjacent block to a memory address A is defined as follows: it is the block corresponding to the memory address A+ block_size.

For example, if a memory address 0x40 misses in the cache and the block size is 4 bytes, then the prefetcher would bring the block corresponding to 0x40 + 4 into the cache.

The prefetcher is activated only on misses and it is not active on a cache hit. If the prefetched block is already in the cache, it does not read the block from memory. With respect to cache replacement policies, if the prefetched block hits in the cache, the line replacement policy status should not be updated. Otherwise, it is treated similar to a block that missed the cache.

Sample Run

Your program should print out the number of memory reads (per cache block), memory writes (per cache block), cache hits, and cache misses for normal cache and the cache with prefetcher. You should follow the exact same format shown below (pay attention to case sensitivity of the letters), otherwise, the autograder can not grade your program properly.

$./first 32 assoc:2 fifo 4 trace2.txt

```
no-prefetch
Memory reads: 3499
Memory writes: 2861
Cache hits: 6501
Cache misses: 3499
with-prefetch
Memory reads: 3521
Memory writes: 2861
Cache hits: 8124
Cache misses: 1876
```

In this example above, we are simulating 2-way set associate cache of size 32 bytes. Each cache block is 4 bytes. The trace file name is “trace2.txt”.
As you can see, the simulator should simulate both catch types with the prefetcher and without the prefetcher in a **single** run and display the results for both.

Simulation Details

1. (a) When your program starts, there is nothing in the cache. So, all cache lines are empty (invalid).

(b) you can assume that the memory size is 2^{48} . Therefore, memory addresses are 48 bit (zero extend the addresses in the trace file if they’re less than 48-bit in length).

(c) the number of bits in the tag, cache address, and byte address are determined by the cache size and the block size;

(d) Your simulator should simulate the operation of a cache according to the given parameters for the given trace

2. For a write-through cache, there is the question of what should happen in case of a write miss. In this assignment, the assumption is that the block is first read from memory (one read memory), and then followed by a memory write.

3- You do not need to simulate the memory in this assignment. Because, the traces doesn’t contain any information on “data values” transferred between the memory and the caches.

4. You have to compile your program with the following flags:

-Wall -Werror -fsanitize=address

Extra Credit (50 points):

As an extra credit, you should implement LRU (Least Recently Used) cache policy. Your program should output exactly the same format output as it shown before. Please note that, you should clearly mention in the report that you've done extra credit otherwise you may not get the points.
Here is an example of running your program with LRU policy.

$./first 32 assoc:2 lru 4 trace2.txt

```
no-prefetch
Memory reads: 3292
Memory writes: 2861
Cache hits: 6708
Cache misses: 3292
with-prefetch
Memory reads: 3315
Memory writes: 2861
Cache hits: 8331
Cache misses: 1669
```

Submission

You have to e-submit the assignment using Sakai . Put all files (source code + Makefile + report.pdf) into a directory named first, which itself is a sub-directory under pa4 . Then, create a tar file (follow the instructions in the previous assignments to create the tar file). Your submission should be only a tar file named pa4.tar. You have to e-submit the assignment using Sakai.

Your submission should be a tar file named pa4.tar. To create this file, put everything that you are submitting into a directory named pa4. Then, cd into the directory containing pa4 (that is, pa4's parent directory) and run the following command:

$tar cvf pa4.tar pa4

To check that you have correctly created the tar file, you should copy it (pa4.tar) into an empty directory and run the following command:

$tar xvf pa4.tar

This should create a directory named pa4 in the (previously) empty directory. Your pa4 folder should contain the following:

- **first**: folder
    - **source code**: all source code files necessary for building your programs. Your code should contain at least two files: first.c and first.h. It should contain code for regular credit and extra credit (if you attempt it).
    - **Makefile**: There should be at least two rules in your Makefile:
        - first: build the executables (first).
        - clean: prepare for rebuilding from scratch.
    - **report.pdf** : In your report you should briefly describe the main data structures being used in your program. More importantly, you should report your observation on how the prefetcher changed the cache hits and number of memory reads. Explain why?

Autograder

First mode
Testing when you are writing code with a pa4 folder.

1. Lets say you have a pa4 folder with the directory structure as described in the assignment.
2. Copy the folder to the directory of the autograder
3. Run the autograder with the following command

$python auto_grader.py

It will run the test cases and print your scores.

Second mode

This mode is to test your final submission (i.e, pa4.tar)
1. Copy pa4.tar to the autograder directory
2. Run the autograder with pa4.tar as the argument as below:

$python auto_grader.py pa4.tar

Grading guidelines

1. We should be able build your program by just running make.
2. Your program should follow the format specified above for the usage interface.
3. Your program should strictly follow the input and output specifications mentioned above. (Note: This is perhaps the most important guideline: failing to follow it might result in you losing all or most of your points for this assignment. Make sure your program's output format is exactly as specified. Any deviation will cause the automated grader to mark your output as "incorrect". REQUESTS FOR RE-EVALUATIONS OF PROGRAMS REJECTED DUE TO IMPROPER FORMAT WILL NOT BE ENTERTAINED.)