CS 211 - Computer Architecture

Instructor: Prof. Santosh Nagarakatte
Programming Assignment 03
Caching

November 28, 2013
Due by December 16, 2013 - 5 pm

1 Overview

The Programming Assignment 03 (PA3) is designed to help you really understand how caches work. Be careful! It will require a substantial implementation effort as well as time and effort to explore and analyze cache behaviors. The usual warning goes double for this assignment: *do not procrastinate*. The programs have to run on iLab machines.

2 Cache Simulator

You will implement a Cache Simulator to evaulate different configurations of Caches, running it on different traces files. You will simulate 3 levels of caches: L1, L2, and L3. All the caches are inclusive. All caches use the same associativity and the replacement algorithm.The sizes of the caches, blocksize and associativity are parameterizable.

2.1 Invocation Interface

Implement a program **cache-sim** that will simulate the operation of a cache. Your program sim should support the following usage interface:

```
cache-sim [-h] -l1size <L1 size> -l1assoc <L1 assoc> -l2size <L2 size>
-l2assoc <L2 assoc> -l3size <L3 size> -l3assoc <L3 assoc> <block size> <replace alg>
<trace file>
```

where:

- $< L1size > / < L2size > / < L3size >$ is the total size of the cache. This should be a power of 2. Also, it should always be true that $< cachesize >$ = number of sets $\times < setsize > \times < blocksize >$. For direct-mapped caches, $< setsize >$ = 1. For $n - way$ associative caches, $< setsize > = n$. Given the above formula, together with $< cachesize >$, $< setsize >$, and $< blocksize >$, you can always compute the number of sets in your cache.
- $< L1assoc > / < L2assoc > / < L3assoc >$ is one of:
  - **direct** - simulate a direct mapped cache.
  - **assoc** - simulate a fully associative cache.
  - **assoc:n** - simulate an $n - way$ associative cache.

- $< blocksize >$ is an power of 2 integer that specifies the size of the cache block.
- $< replacealg >$ is one of:
  - **FIFO** - simulate a first-in-first-out replacement algorithm.
  - **LRU** - simulate an LRU replacement algorithm.
- $< tracefile >$ is the name of a file that contains a memory access traces (Each line has a 64-bit hexadecimal address similar to Programming Assignment 1).

If $-h$ is given as an argument, your program should just print out help for how a user can run the program and then quit.

Erroneous inputs should cause your program to print out:

```
ERROR: <an informative error message>
```

Otherwise, your program should print out the number of memory accesses, cache hits, and cache misses. You should follow the format of this example:

```
Memory accesses: 1553
L1 Cache hits: 1400
L1 Cache miss: 453
L2 Cache hits: 100
L2 Cache miss: 100
L3 Cache hits: 100
L3 Cache miss: 100
L1 Cold misses: 53
L2 Cold misses: 53
L3 Cold misses: 53
L1 Conflict misses: 100
L1 Capacity misses: 300
```

2.2 Simulation Details

At the start of your simulation - that is, when your program first starts running - all entries in the cache should be invalid. The number of bits in the tag, cache address, and byte address are then determined by the cache size, cache associativity, and block size.

Your simulator should simulate the operation of a cache according to the given parameters for the given trace. At the end, it should print out the number of memory accesses, cache hits, and cache misses as mentioned above.

3 Analysis

Pretend you are designing a new computer. Use your simulator, the memory traces given, and assumptions about cache and memory performance vs. cache and memory dollar cost to design the data cache for your machine.

- Assume that you will only have an L1 cache. What would be the appropriate parameters (e.g., size, block size, associativity, etc.) for your cache and given workload to attain 95% hit rate?
- Assume that you will only have an L1 cache. What would be the appropriate parameters (e.g., size, block size, associativity, etc.) for your cache and given workload to attain 99% hit rate?
- Repeat the above analysis (that is, design the data cache subsystem) assuming that you can have both an L1 and L2 cache. What would be the appropriate size for an L2 cache to attain 95% hit rate for L1 and L2 cache?

4 Submission

You have to e-submit the assignment using Sakai. Your submission should be a tar file named **pa3.tar** that can be extracted using the command:

```
tar -xf pa3.tar
```

Extracting your tar file must give a directory called **pa3**. This directory should contain:

- A sub-directory named **docs**. docs must contain:
  - **readme.pdf**: this file should describe the design and implementation of your cache simulator. You should think about questions such as: How did you implement the various replacement algorithms? How did you account for the different types of misses? For this assignment, you do not have to worry about analyzing the time and space behavior of your program.
  - **analysis.pdf**: this file should contain your analysis for Section 3.
- A sub-directory named simulator. simulator must contain:
  - **Makefile**: there should be at least two rules in your Makefile:
    - **cache-sim** build your sim executable.
    - **clean** prepare for rebuilding from scratch.
  - **source code**: all source code files necessary for building sim. At minimum, this should include two files, sim.h and sim.c. This assignment is large enough that you might consider splitting your code across multiple files.

We will compile and test your programs on the iLab machines so you should make sure that your programs compile and run correctly on these machines. You must compile all C code using the gcc compiler with the $-ansi$ $-pedantic$ $-Wall$ $-m32$ flags.

5 Grading Guidelines

5.1 Functionality

The most significant part of your grade will be based on programmatic checking of your program. That is, we will build a binary using the Makefile and source code that you submitted, and then test the binary for correct functionality against a set of inputs. Thus:

- You should make sure that we can build your program by just running make.
- You should test your code as thoroughly as you can. In particular, your code should be adept at handling exceptional cases.

Be careful to follow all instructions. If something doesn't seem right, ask.

5.2 Design

Having said the above about functionality, design is a critical part of any programming exercise.

In particular, we expect you to write reasonably efficient code based on reasonably performing algorithms and data structures. Thus, the explanation and discussion of your design and implementation in the readme.pdf files will comprise a non-trivial part of your grade. Give careful thoughts to your writing of this file, rather than writing whatever comes to your mind in the last few minutes before the assignment is due.

5.3 Coding Style

Finally, it is important that you write "good" code. Unfortunately, we won't be able to look at your code as closely as we would like to give you good feedback. Nevertheless, a part of your grade will depend on the quality of your code. Here are some guidelines for what we consider to be good:

- Your code is modularized. That is, your code is split into pieces that make sense, where the pieces are neither too small nor too big.
- Your code is well documented with comments. This does not mean that you should comment every line of code. Common practice is to document each function (the parameters it takes as input, the results produced, any side-effects, and the function's functionality) and add comments in the code where it will help another programmer figure out what is going on.
- You use variable names that have some meaning (rather than cryptic names like i).

Further, you should observe the following protocols to make it easier for us to look at your code:

- Define prototypes for all functions.
- Place all $prototype$, $typedef$, and $struct$ definitions in header ($.h$) files.
- Error and warning messages should be printed to stderr using fprintf.

6 Memory Access Traces

We have provided 3 memory traces with the assignment. Each line is a trace file is a 64-bit hexadecimal address. As with the other programming assignments, your program is required to produce expected outputs on other traces and produce the correct results.