Project 2: Cache Simulation

CMPEN 431: Introduction to Computer Architecture

Fall 2022: Prof. Kiwan Maeng

Due Date: 2nd December 2022 (11.59 PM) EST

1. Project Goal

The course projects in this class serve to explore the microarchitectural design space of uniprocessor design parameters. This project will help you to

a) learn to read the provided framework code

b) see how a simple cache simulation model can be developed

c) once fully functional, explore the marginal benefits of different cache hierarchies to help develop the future project

2. Instructions

1) Copy the tarball to a CSE machine and extract it into a local directory of your choosing. You **should** use CSE Linux Lab machines (i.e. e5-cse-135-01.cse.psu.edu through e5-cse-135-40.cse.psu.edu). To connect to these machines remotely, you should install a VPN on your computer and then use ssh to connect to one of the machines. For more details, please refer to the CSE Student Lab access information: https://www.eecs.psu.edu/cse-student-lab-access/index.aspx.

This project simulates putting the memory references and values generated through naive ($O(N^3)$) matrix multiplication of square matrices of size N through a parameterized cache hierarchy, that is, a specific cache will be generated respectively by the arguments in each test case in Makefile. The arguments represent as below:

1st, the size of the simulated matrix
2nd, the number of matrix multiplications to perform
3rd, the name of the cache level (for example, the second-level L2)
4th, the size of this level
5th, the associativity of this level
6th, the block size of this level
7th and the subsequent, the first-level L1 cache, and its arguments.

If you implement it correctly, your logic would work for all these test cases. Most of the functionality for this program has already been provided. However, five key functions needed to properly perform caching (setSizesOffsetsAndMaskFields, getindex, gettag, writeback, fill) are currently implemented as stub functions that either does nothing, causing the program to

crash if they are relied upon. Your job will be to implement these missing functionalities within the functions defined in "YOURCODEHERE.c", and descriptions of the functionality of each function are in YOURCODEHERE.h.

You will need to read through the provided framework to figure out how to properly use the "performaccess function to set local contents based on another level's data (fill) and to write data from the local contents into the next level of the memory hierarchy (writeback).

You will need to familiarize yourself with the existing functions defined in csim.c, specifically "performaccess", and the cache structure defined in csim.h, **although you are not allowed to modify them.** You are likewise not allowed to modify anything other than implementing the missing functionalities within the functions defined in "YOURCODEHERE.c". You will invoke "performaccess" in your logistic, the input argument “size” could be fixed as 8.

Your project, once complete, will be able to correctly execute all tests invoked by "make test" as well as other cache and matrix configurations not present in the test list. The test list was already included in the Makefile. Only cache hierarchies with monotonically nondecreasing block sizes (in integer multiples of **8 bytes**) throughout the cache hierarchy will be tested. Similarly, only cache hierarchies with monotonically nondecreasing capacity from upper to lower caches will be tested.

2) Ensure that your environment is correctly configured (e.g. with default gcc, etc.) by running "make test". You can verify the correct initial state of your environment/files by noting the following:

a) the code should compile without any errors or warnings.

b) the first test case (no cache instantiated) should run to completion and match the output in the included copy of the output from running “make test” on a completed version of the program

c) the second test case should quickly generate a “Segmentation fault” due to the unimplemented stub functions

3) Modify YOURCODEHERE.c -- this is the **only** file you will be modifying and turning in. Your project **MUST** compile without modification to the Makefile, or any other source files. Your code will be recompiled against the other files in their original state, **on CSE servers**. Any reliance on additional modifications will likely result in non-compiling status under test and a **zero** for the project. Please note that the CSE lab machines are 64-bits (represented as *localVAbits* variable), so 1 word = 8 bytes. You will reflect this in your implementation. Please ensure that any code you develop on a non-CSE platform works on the CSE servers, as the code is **NOT GENERALLY PORTABLE.**

4) Continue to test your project. All tests in "make test" should run to completion (expected total run time 1-2 minutes, mostly in the last test). Statistics from your output file (NMM-

csim.testout) for matrix sizes <= N=8 should match the provided output statistics(NMM-csim.WORKING.testout) exactly. Statistics for larger matrix sizes should be very similar but the output may or may not be identical.

3. Your Submission

You will turn in only the "YOURCODEHERE.c" file via CANVAS. Your results will be tested on CSE lab machines. In addition to your code fixes, provide, as a multi-line comment in YOURCODEHERE.c, a description of your testing approach to check whether or not your output is correct, given that the contents of invalid memory locations within the cache will contain arbitrary data that may not match between different runs.