CSCE 231/ 2303 Spring 2020 Assignment 6: Cache Simulation and Analysis

Assigned: Thursday, April 9th

Due: Thursday, April 16th

Delayed submission with penalty until Saturday, April 18th at mid-night.

Goals

This assignment is an individual assignment and you will work on it on your own. The goal of this assignment is to write a cache simulator in C/C++ and to experiment different cache parameters on running different workloads. Moreover, you are required to model your experiment results in a comparative way using multiple graphs that shows the impact of changing cache size, cache write policy, and different program memory access profiles and data sizes.

Details

You are required to write a Direct Map Cache Simulator. Luckily, you will not write the simulator from scratch, rather you will be provided with a C/C++ skeleton for the cache simulator, and you are required to amend the skeleton with the cache Direct Map logic. You will be precisely provided by a code tree, main class hierarchy skeleton, some parts of the code and a make file. You will also be provided by a sample programs that you will initially use to test your cache simulator and then change it in different ways to apply different locality setups and data sizes.

Your simulator should be designed to receive a trace file for memory accesses performed by a program. The trace file contains a line for each memory access which contains the instruction memory address, followed by type of the access; Read (R) or Write (W), and finally the memory address accessed by that instruction. Here is a sample of an input file that the simulator should accept:

```
0x7f9643ee9ee8: W 0x7f96441109f0
0x7f9643ee9f4b: W 0x7f9644110aa0
0x7f9643ee9f53: R 0x7f964410fe78
0x7f9643ee9f4b: W 0x7f9644110a50
0x7f9643ee9f53: R 0x7f964410fe88
0x7f9643eea333: W 0x7f9644110c88
0x7f9643ee9f53: R 0x7f964410fe98
0x7f9643ee9f4b: W 0x7f9644110a58
0x7f9643ee9f53: R 0x7f964410fea8
0x7f9643ee9f4b: W 0x7f9644110a60
0x7f9643ee9f53: R 0x7f964410feb8
```

The question now is how to generate the simulator input file from a running program? Luckily, Intel has a dynamic binary instrumentation profiling tool that can achieve that; generate a file with the above format for any program run. The Intel tool is called the **PinTool** and can be downloaded from the link https://software.intel.com/en-us/articles/pin-a-binary-instrumentation-tool-downloads. As you are expected to implement this project totally on Linux, you can also directly download the tool through this link https://software.intel.com/sites/landingpage/pintool/downloads/pin-3.11-97998-g7ecce2dac-gcc-linux.tar.gz. Also, a full documentation for the tool is available at https://software.intel.com/sites/landingpage/pintool/docs/81205/Pin/html/.

Download the tool into some directory and perform the following steps:

```
wget https://software.intel.com/sites/landingpage/pintool/downloads/pin-3.11-97998-g7ecce2dac-gcc-linux.tar.gz

tar -xvzf pin-3.11-97998-g7ecce2dac-gcc-linux.tar.gz

cd pin-3.11-97998-g7ecce2dac-gcc-linux/source/tools/

Make all
```

Now you can use the tool to run any program and generate a cache trace file as follows:

```
cd pin-3.11-97998-g7ecce2dac-gcc-linux/
./pin -t ./source/tools/ManualExamples/obj-intel64/pinatrace.so -- ls
```

The above command runs the “ls” program that lists the content of a directory and logs all the memory accesses into a file called pinatrace.out. You should rename the file before running the pintool again so you do not overwrite it.

Now your simulator needs to be as configurable as possible. Your simulator is a command line tool that should receive the following command line parameters:

```
./bin/cache_sim <cache mode: wb|wt> <file_name> <cache-size> <line-size> <cache-read-cycles> <cache-write-cycles> <mem-read-cycles> <mem-write-cycles>
```

The parameters, following the above order strictly are:

- **cache mode:** this is the cache update mode which can be either write back (wb), or write through (wt).
- **file_name:** memory trace file name generated by the pintool.
- **Cache-size:** the number of lines in the cache.
- **Line-size:** the size of a cache line in bytes.
- **Cache-read-cycles:** number of CPU cycles spent to read from the cache.
- **Cache-write-cycles:** number of CPU cycles spent to write from the cache.

- **Memory-read-cycles:** number of CPU cycles spent to read from the memory.
- **Memory-write-cycles:** number of CPU cycles spent to write from the memory.

Here is an example on how to run your simulator:

```
./bin/cache_sim wb ./pintrace.out 1024 2048 2 4 8 16
```

Your simulator output should be at least the following:

- Total Misses.
- Total Hits.
- Memory Read Access Attempts.
- Memory Write Access Attempts.
- Total Memory Access Attempts.
- Memory Read Access.
- Memory Write Access.
- Total # of cycles for cache read.
- Total # of cycles for cache writes.
- Total # of cycles for cache access.
- Total # of cycles for memory read.
- Total # of cycles for memory writes.
- Total # of cycles for memory access.

You are required to use the sample programs and modify it such that you cover a lot of cases, in a way to show the effect of spatial locality on the cache performance as well as size of data being processed. You should create different variations of the program. You are also more than welcome to write other programs that adopt different memory access workloads to assess their effect on the Direct Map cache model, such as temporal locality. In all cases you will be running the simulator against memory traces generated by the PinTool. You should build as many comparative graphs as you can to compare the effect of changing different parameters. The more and fine grained comparison, the better your grade will be. You should have a detailed writeup that explains all the graphs you present and your interpretation and analysis of the results.

What to submit

1. Your full in-line documented source code of the simulator.
2. Readme file to state any assumptions about your simulator and includes a How-To for running the simulator and reproduce your results.
3. A PDF report documents the results, present graphs, as well as your interpretation and analysis.
4. Your code and experiments should run on Linux.

How to submit:

Compress all your work: source code, report, readme file, and any extra information into a zip archive. You should name your archive in the specific format <Student_ID>_<Name>_Assignment6.zip. Finally, upload your code to blackboard.

Grade

This assignment is worth 10% of the overall course grade. The assignment will be graded on a 100% grade scale, and then will be scaled down to the 10% its worth. The grading of the assignment will be broken down as follows:

1. 10 % for just submitting a meaningful assignment before or on the due date. This 10% does not account for the correctness of your assignment but submitting an empty assignment without code will definitely results in loosing this 10% and consequently the whole grade of this assignment.
2. 25 % for the correctness and the quality of your code.
3. 10 % inline documentation.
4. 55 % your PDF report and the types of experiments you conducted.

Very Important Note: The 10% contribution of this assignment to your overall grade might change (most probably increase) due to the current situation.

Delays

You have up to 2 working days of delay, after which the assignment will not be accepted and your grade in that case will be ZERO. For every day (of the 2 allowed days), a penalty of 10% will be deducted from the grade. And of course you will lose the 10% mentioned in point 1 above under the “Grade” section.