Project 4. Cache Simulator

Due 23:59, June 2nd, 2022 (KST)

TA: Minsu Kim, Sanghyeon Lee

1. Introduction

This fourth project is to build a trace-based cache simulator. The cache should be configurable to adjust capacity, associativity, and block size with command-line options. **Please read this document carefully before you start.**

****If you have any questions related to the project, please ask them on the Piazza board or email(cs311_ta@casys.kaist.ac.kr) or office hour.**

2. GitLab Repository

Basically, you just need to repeat the same procedure of forking and cloning the repository explained in the document for previous projects. The only difference is that you should start from the TA's repository of **Project 4**. (http://cs311_2022.kaist.ac.kr/root/project4-cache-simulator) Please check prior notifications available in Piazza board, for more information about getting started with GitLab.

3. Simulator Specification

The simulator gets a trace file as an input. The input trace file contains a sequence of read or write operations with addresses. For each step, an entry of the trace is fed to the cache simulator, to simulate the internal operation of the cache. The write policy of the cache must be *write-allocate* and *write-back*. The replacement policy must be the perfect LRU.

Cache Parameters

The cache simulator has three configurable parameters: capacity, associativity, and block size. Range of each parameter is specified below.

- Capacity: 4B (one word) ~ 8KB
- Associativity: 1 way ~ 16 ways
- Block size: 4B ~ 32B

Capacity and block size should be specified with byte granularity, and all three parameters should be power of two.

```
Ex) Capacity 4KB, Associativity 4-ways, Block size 32B -> 4096:4:32
```

Input Trace File

An input trace file contains a sequence of read or write operations with addresses like below.

```
…
R 0x10000000
W 0x10003231
R 0x12341245
R 0x10003231
W 0x10023414
…
```

R denotes read, while W means write.

Output

Simulator output must contain the following statistics:

- the number of total reads
- the number of total writes
- the number of write-backs
- the number of read hits
- the number of write hits
- the number of read misses
- the number of write misses

The order of output results is Cache Configuration -> Cache Statistics -> Cache Content (if –x option is on). Please check sample outputs for given examples for more information.

4. Simulator Options

Basic command

```
$ ./cs311cache -c capacity:associativity:block_size [-x] <input trace file>
```

Options

- c: Cache configuration specification.
- x: Dump the cache content at the end of simulation.

**** Cache Content ****

Cache content is not a data content, but the address which is aligned with block size.
For example, assume block size is 16B (2^4 B). When the address 0x10001234 is stored in cache, the content of the cache entry is 0x10001230 since block size bits (4 bits) are masked by 0.

Tag bit	Index bit	Block offset (0)

5. Grading Policy

Basically, grading policy is similar to prior projects. Submissions will be graded based on the 4 examples**: 3 given examples in 'sample_input' directory, and 1 hidden example of which inputs and outputs are both hidden**. The hidden example has similar complexity with respect to given examples.

To get scores, your simulator should print the exactly same output as the reference solution. You can evaluate your code within given examples by comparing your output with files in 'sample_output' directory. **You may check the correctness of your code by executing** `make test` **at your working directory of this project**.

If there are any differences (including whitespaces) it will print the differences. If there is no difference for an example, it will print "Test seems correct".

Please make sure your outputs for given examples are **identical to the files in the sample_output directory, without any redundant prints. Every single character of the output must be identical** to the given sample output. Otherwise, you will receive **0 score** for the example.

Note that **simply hard-coding outputs for given examples would lead you to 0 score** for this project.

6. Submission (Important!!)

Make sure your code works well on our class Linux server.
It is highly recommended to work on the class server, since your project will be graded on the same environment as those servers.

Add the 'submit' tag to your final commit and push your work to the gitlab server. If there is no "submit" tag, your work will not be graded. Please do not forget to submit your work with the tag.

Please make sure you push your work before the deadline. If you do not "push" your work, we won't be able to see your work so that your work will be marked as unsubmitted.

For more information about submission procedure, please refer to the specification of project 1, chapter 5 'Submission'.

7. Late Policy & Plagiarism Penalty (Important!!)

You will lose **30%** of your score on the **first day** (June 3rd 0:00~23:59). We will **not accept** any works that are submitted after then.

Be aware of plagiarism! Although it is encouraged to discuss with others and refer to extra materials, **copying other students' or opened code is strictly banned: Not only for main routine functions, but also helper functions.**

TAs will compare your source code with open-source codes and other students' code. If you are caught, you will receive a serious penalty for plagiarism as announced in the first lecture of this course.

If you have any requests or questions regarding administrative issues (such as late submission due to an unfortunate accident, GitLab is not working) please send an e-mail to TAs.

8. Tips

- Modifying your repository via http may cause some problems.
- In this project, you can freely define functions, macros, structs and classes (C++) you need.
- **You may use C++ instead of C if you want.** Just make sure your '`make test`' command works properly.
- **Please start as early as possible**.
- Please **read this document and given skeleton code carefully before you start.**