Program Assignment 4 Cache Simulation

Dec 18, 2018

By Jing-Jia Liou

Contents

1 Problems

- In this homework, there are two parts. In the first part (70%), we will implement the LRU policy in a cache simulator. In the second part (30%), we will design and optimize a cache architecture.

1.1 Penalty of plagiarism

- Each time you submit a plagiarized code, your grade of the homework will be discounted to 90%. The check is done in batch every hour. The check is based on code similarity at a 30% threshold (meaning 30% of your code is structurally identical to other's codes).
- The deduction will be **accumulated each time** you submit a plagiarized code. Please do not use trial-and-error approach to adjust your codes.
- If your final version (last submission before due date) is a plagiarized code, no credit will be given.
- If your simulation reports is not identical to your program output, your code'll be considered as plagiarism.

1.2 Part I LRU replacement policy

Due: 11:59 am on Jan. 3, 2019

Proportion: 70%

- Please implement a LRU replacement policy in our cache simulator. Please read our lecture notes about LRU policy. We will check the correctness of your implementation with following tables.
- We prepared a C++ program template as a starting point. Please download codes by the following command:

```
$ git clone http://gitlab.larc-nthu.net/ee3450_2018/pa4_template.git pa4
```

- Compiling the simulator

  The program directory structure looks like this

```
pa4/
|— CMakeLists.txt
|— src/
|   |— CMakeLists.txt
|   |— cache.cpp
|   |— cache.hpp
|   |— main.cpp
```

  Here we use **cmake** instead of traditional makefile to build your project, you can add any C++ source files in **src**directory and cmake will detect them automatically.
  Use the following commands to build your simulator (Assume you are already in the project root folder):

```
$ mkdir build && cd build
$ cmake ..
$ make
```

  Then the binary executable file will be generated and named as **cache_sim**.
- We also prepared a few trace files under ~ee345000/pa4/trace/ directory. You can use the following command to copy them into your project directory.

```
$ cp -r ~ee345000/pa4/trace .
```

Trace files are load/store records dumped from benchmark programs. We will use trace files to represent program memory accesses. The format of a trace file is as follows:

```
l 0x1fffff50
```

l or s means load or store.

0x1fffff50 is the 32-bit address in hexadecimal, note that the block size should be at least 4B.

- There are three sample config files under config/ directory. Config file is used to specify a cache architecture:

```
256 # Cache size: 256KB
8   # Cache block size: 8B
1   # Associativity: Direct mapped
64  # Cache size: 64KB
32  # Cache block size: 32B
2   # Set associative
4   # Number of sets
1   # Replacement policy: 1. Random 2. LRU
8   # Cache size: 8KB
64  # Cache block size: 64B
3   # Full associative
1   # Replacement policy: 1. Random 2. LRU
```

- Run simulation and dump your simulation results as a file

  Assume you are running with "gcc.trace" and "cache1.cfg". Use the following command to dump the simulation results into text file.

```
$ ./cache_sim ../trace/gcc.trace ../config/cache1.cfg > gcc.txt
```

- Your simulator output will be as follows:

```
Test file: ../trace/gcc.trace
```

```
Cache size: 256KB
Cache block size: 8B
Associativity: direct_mapped
Replacement policy: None

Number of cache access： 515683
Number of cache load： 318197
Number of cache store： 197486
Cache hit rate: 0.958347
```

- To verify your LRU implementation, please use following tables to check your outputs. You can also find results in results/ directory.

Test file	gcc	gzip	mcf	swim	twolf
Total inst.	515683	481044	727230	303193	482824
Load	318197	320441	5972	220668	351403
Store	197486	160603	721258	82525	131421

- 256KB, 8 Bytes/line, direct_mapped, None

Test file	gcc	gzip	mcf	swim	twolf
Hit Rate	0.958347	0.667072	0.010379	0.934319	0.988443

- 64KB, 32 Bytes/line, 4-way set_associative, LRU

Test file	gcc	gzip	mcf	swim	twolf
Hit Rate	0.987636	0.668253	0.752378	0.978618	0.996578

- 8KB, 64 Bytes/line, fully_associative, LRU

Test file	gcc	gzip	mcf	swim	twolf
Hit Rate	0.989703	0.668319	0.876024	0.986652	0.997067

- Note that your simulation results should match above tables or have better hit rates for the same cache architecture and trace files.
- Please run all 5 different trace files with cache2.cfg and cache3.cfg and dump the results.

1.3 Part II Cache Optimization

Due: 23:59 on Jan. 6, 2019

Proportion: 30%

- In this part, we will design an optimized cache architecture with a cache size of 64KB. The design parameters are cache block size, number of ways, and replacement policy. Our objective is to have an average miss rate as low as possible under the 64KB constraint over several trace files.
- If two cache architecture has similar miss rates (difference < 0.1%), we will use hardware cost to find the best cache design. The hardware cost is basically the total memory bits used in the cache. Note that we will actually use a SRAM simulator (CACTI) to estimate the total memory area (not just bit numbers).
- Cache designs from all students will be ranked in normal distribution percentile for grading.
- Note that you may use other replacement policy to reduce miss rates instead of just LRU or Random.
- To evaluate the hardware cost, we prepare an interactive evaluation tool, you can use it by the following command.

```
$ ~ee345000/pa4/bin/eval
```

The tool will ask for your cache configuration and generate a hardware score for your cache design. The terminal output will looks like the following. In this case, the score of the design is 86.37. Note that the higher score, your cache needs less hardware (better).

```
Copy tech_params to local successfully!

Block size(bytes): 4

Associativity(2^n with n >= 0, e.g., 1, 2, 4...): 4

Generated 'cacti.cfg' successfully!

Final score: 86.37

$
```

Also note that this tool will generate a CACTI configuration file named **cacti.cfg** in current folder. You will need to submit this configuration file, too.

2 Submission

2.1 Part I

- Files to submit

1. Project directory (pa4): Including all C++ source files and CMakeLists.txt, but **DO NOT** submit the build directory and trace files.
   - The detailed file structure is listed below.

```
pa4/
|— CMakeLists.txt
|— src/
|   |— CMakeLists.txt
|   |— cache.cpp
|   |— cache.hpp
|   |— main.cpp
|   |— any .cpp files you created
```

1. Simulation results: Please create a directory named as **result** and put all the result files inside. Name your results according to trace and config files. For example, if you run gcc.trace with cache2.cfg, the result file must be gcc_cache2.txt.
   - The detailed file structure is listed below.

```
result/
|— gcc_cache2.txt
|— gcc_cache3.txt
|— gzip_cache2.txt
|— gzip_cache3.txt
|— mcf_cache2.txt
|— mcf_cache3.txt
|— swim_cache2.txt
|— swim_cache3.txt
```

```
|— twolf_cache2.txt
|— twolf_cache3.txt
```

- If your student ID is **103061232** (as an example),
  1. Zip your project directory as hw4_1_103061232.zip.
  2. Zip the simulation report directories as hw4_1_103061232_report.zip.
  3. Submit them via the link

2.2 Part II

- Files to submit:

  1. Project directory (pa4_opt): including all C++ source files, optimized cache configuration file and CMakeLists.txt. Please **DO NOT** submit the build directory and trace files. They are not necessary and too big to transfer/store at our gitlab site.
     - Please name you optimal configuration file as cache_opt.cfg
     - The detailed file structure is listed below.

```
pa4_opt/
|— CMakeLists.txt
|— src/
|   |— CMakeLists.txt
|   |— cache.cpp
|   |— cache.hpp
|   |— main.cpp
|   |— any .cpp files you created
|— config/
|   |— cache_opt.cfg
```

  1. Simulation results: Please create a directory named as **result_opt**. Then put all the five results (use your optimized cache configuration) and CACTI configuration file (cacti.cfg) in the directory.
     - The detailed file structure is listed below.

```
result_opt/
|— gcc.txt
|— gzip.txt
```

- |— mcf.txt
- |— swim.txt
- |— twolf.txt
- |— cacti.cfg

1. (Optional) Readme file: If you implement some extra feature in the simulator, please write a simple readme file in Markdown format.

- If your student ID is **103061232** (as an example),
  1. Zip your project directory as hw4_2_103061232.zip.
  2. Zip the simulation report directories as hw4_2_103061232_report.zip.
  3. Readme file as hw4_2_103061232.md
  4. Submit them via the link1

3 Grading

- No credit for dead or crashed codes. (We'll use DEBUG mode specified in CMakeLists.txt to build your program)
- No credit for codes with wrong output formats.
- Each simulation (open traces) should be finished in 15 seconds. You may use "time" command in Linux to find out run times.
- Part I
  - The hit rate of each simulation should be identical to the tables listed (or better). If some value mismatches the table, only partial credits will be given.
- Part II
  - We will use miss rate and hardware cost to rank all results in normal percentile. Students who achieve the lowest miss rate (take average of all traces) with the lowest cost will get the highest score.
  - Note that we will test your cache design with some hidden traces.
  - If implement extra features, we'll give you extra credits (even if your results do not rank high in above evaluation). Please specify clearly in your README file.