Advance Computer Architecture
Project 1: L2 Cache Replacement Policies

Due:11:59PM, 12/1

1 Project description

In this project, you are going to study and implement several cache replacement policies on **Simplescalar**, a single-core processor simulation toolkit. We provide you 11 SPEC2000 CPU benchmark programs to perform basic evaluation as a baseline and compare the baseline with other replacement policies. You are able to (1) simulate CPU running these benchmark programs with different built-in hardware configurations by editing provided configuration file **default.baseline.cfg**, or (2) design your customized configuration by modifying the source code of Simplescalar and run the simulation.

We enumerate three L2 cache replacement policies required by this project:

1. **Default LRU** (built-in in Simplescalar)
   (a) No source code modification required.
   (b) Trace the source code and answer the question: what kind of LRU replacement policy was implemented in Simplescalar?
2. **Not Recently Used** (NRU, the single-bit version)
   (a) NRU policy was also described in Jaleel et al.'s RRIP paper in ISCA 2010.
   (b) Replacement scanning starts from the head.
   (c) NRU is a degenerated case of RRIP. Please refer to the paper for details.
3. **Dynamic Re-reference Interval Prediction on Set Dueling** (DRRIP)
   (a) Implement the DRRIP policy by Jaleel et al.'s paper in ISCA 2010. You may also need to understand the **DIP-SD** method described by Qureshi et al. in ISCA 2007.
   (b) There are 32 dedicated SRRIP sets and 32 dedicated BRRIP sets, along with the remaining sets which are followers. Please employ the following hash functions in your implementation for cache configurations.
      i. An SRRIP set iff (**cache_set_ID**) mod ($\frac{total\#ofsets}{32}$) == 0
      ii. A BRRIP set iff (**cache_set_ID+1**) mod ($\frac{total\#ofsets}{32}$) == 0
      iii. The rest are the follower sets when both of the above conditions fail.
   (c) Implement a **3-bit RRPV**(re-reference prediction value) for each cache line, i.e., M=3 as defined in the paper.

(d) Follow the method described in the ISCA 2007 paper for the throttle parameter "ϵ" used in BRRIP policy, that is, to implement a **5-bit BIRCTR** non-saturation counter for BRRIP. Whenever BIPCTR equals to 0, you insert the cache line with *long* re-reference interval, i.e., $2^M - 2$, otherwise, with *distant* re-reference interval, i.e., $2^M - 1$.

(e) Implement a 10-bit PSEL saturation counter for the selection between SRRIP and BRRIP of follower sets.

2 Simulation to carry out

1. For each L2 cache replacement policy, you need to simulate:
   - **256KB 8-way** L2 cache with **64-byte line size** and **10 cycles hit latency**, which is the baseline.
   - **1MB 8-way** L2 cache with **64-byte line size**, with **hit latency set to 14** cycles.
   - Change the associativity to 16-way for both cases above.
2. For each cache configuration, you need to simulate **11 selected benchmark programs**. Please simulate **fast-forwarding 1 billion instructions followed by another 1 billion instruction** simulation for each benchmark program (i.e., -fastfwd 1000000000 -max:inst 1000000000), which is the default knob setting of Simplescalar in the provided makefile, **SPEC2000.make**.
3. Plot simulation results of the following metrics in your report:
   (a) sim_IPC
   (b) ul2.miss_rate
4. The total number of simulation instances in this project = 3 replacement policies × 11 benchmark programs × 2 cache capacities × 2 associativities = 132 simulations

3 Get started with Simplescalar

Simplescalar provides several simulators inlcuding sim-safe, sim-cache, sim-outorder, etc... We will use "**sim-outorder**" in this project. The simulator is capable of executing two ISA: PISA and Alpha. SPEC2000 benchmarks we provided are based on Alpha ISA. The following demonstrates how to compile your simulator for Alpha binaries.

1. Connect to your Linux machine. Though we only tested on Ubuntu and CentOS, any Linux distribution should be ok.
2. Download and extract the archive file **ee6455_p1.tar**:
   ```
   # tar xvf ee6455_p1.tar
   ```
3. Open Simplecalar 3.0 folder and build the simulator and Alpha binaries:
   ```
   # cd simplesim-3.0
   # make config-alpha
   # make sim-outorder
   ```
   Depending on your Linux platform, during the compilation you may experience several warnings or even errors. Most of warnings are negligible. However, if you do encounter errors, in most of the cases, you need to fix some header files in the source codes. For Solaris, you may need to change a syscall name in the syscall.c file.
4. Now you can test your simulator with a specified Alpha binary and a configuration file, by the command:
   ```
   # ./sim-outorder -config ../default.baseline.cfg ./tests-alpha/bin/test-math
   ```
   The simulation result will be reported on your terminal.

5. Most of your design effort should be put into **cache.c** and **cache.h**. This project can be done without modifying any other file. To help TA collect your simulation results more easily, please **DO NOT** change the simulation output format, which should be seen like this:

```
...
...
ul2.miss_rate       0.6423 # miss rate (i.e., misses/ref)
ul2.repl_rate       0.6423 # replacement rate (i.e., repls/ref)
...
...
```

6. To run 11 SPEC2000 benchmark programs, please leave the current directory and execute the provided Makefile, **SPEC2000.make**

```
# cd ..
# make -f SPEC2000.make all
```

or you simply want to run a single benchmark program, you can specify the name of it.

```
# make -f SPEC2000.make mcf
```

Using the above Makefile will output *.stdout and *.stderr in "results" directory for each benchmark program. The simulation result will be stored in ***.stderr** file instead of shown on the terminal.

4 Hints

- In default.baseline.cfg, the line 30 "-cache:dl2 ul2:512:64:8:l" states that your L2 cache has 512 sets, 64-byte line and 8 ways and use LRU policy.
- As mentioned earlier, some simulation knobs are defined in **SPEC2000.make**, which are **-fastfwd 1000000000 -max:inst 1000000000**
  You can change them to any number you prefer to save your time, but don't forget to reset them to 1 billion when doing the final simulation.
- The proper way to debug yuor program is to use debugging tool such as gdb, ddd or eclipse. We suggest you debug your program using test program in the **simplesim-3.0/tests-alpha/bin** directory to begin with.
- sim-outorder models the micro-architecture in reasonable details to report the performance results of the benchmark program simulated, eg., IPC, cache miss rates, branch miss-prediction rates, etc. For this project, it is not necessary to understand the entire simulator. As mentioned earlier, most of your modifications for incorporating different cache replacement policies should be in **cache.c and cache.h**. You also need to **"register" new knobs** to add your policies as new knobs for the simulator. **Please use 'n' and 'd' to represent NRU and DRRIP. We will use these new knobs to test your program**.
- Once you are ready to run all the simulations, re-compile your binary with -O3 compiler option in gcc, which gives a binary with much higher performance.
- Please take the simulation hours into account when working on this project. Depending on the machine you use, to execute 1 billion instructions in sim-outorder may take 10 minutes or more. Given the large number of simulations to complete this project, you want should start as soon as possible. For example, you would like to start the default LRU cases, in which you will do almost nothing but change the config file to generate all the results after you finish tracing the source code.
- Make sure you have built **sim-outorder** and installed **csh** before running SPEC2000.make. For Debian Linux distribution and its variants, you can use the command to install csh:

```
# sudo apt-get install csh
```

- **This project must be done individually, For those who violate the rule, both the originator and the copier will receive zero credit.**

5 Report format

Attached ISCA papers are good templates for us. We strongly suggest you study them before writing the report, which should **at least** include the followings:

- **Implementation details**
  You can enumerate each cache replacement policy and highlight codes you modified or inserted. **Explain the idea behind your modification** rather than simply pasting the code. Don't for get to answer the question: what kind of LRU replacement policy was implemented in Simplescalar?

- **Experimental results and discussion**
  Plot IPC, ul2.miss_rate for each simulation and **analyze the results**. For example, discussing about how do cache size, replacement policy and associativity affect IPC and miss rate. You might further simulate different designs of DRRIP (eg.adjusting throttle parameter, number of RRPV bits, initial value of PSEL) for more analysis. Note that showing both absolute and relative results is more preferable, see below:

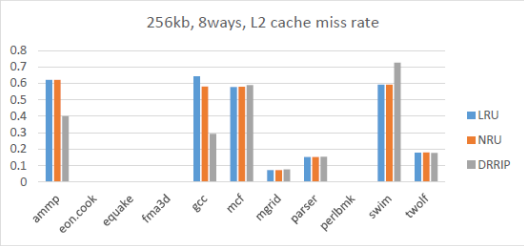

Figure 1: A sample of absolute miss rate plot

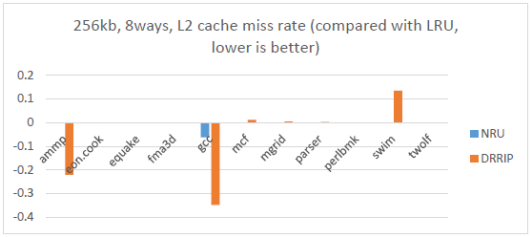

Figure 2: A sample of relative miss rate plot

- **Conclusion**
  Make a brief conclusion on your work.

6 What to submit (important!!!)

- .c and .h files you modified in simplesim-3.0. You don't need to upload those files you never edit.

- A PDF report named YOUR_STUDENT_ID.pdf

- Please use the below command to wrap above files and then upload to iLMS.
  ```
  # tar -czvf YOUR_STUDENT_ID.tar YOUR_REPORT YOUR_CODES
  ```
  For me, I should use:
  ```
  # tar -czvf 103061568.tar 103061568.pdf cache.h cache.c
  ```
  And then upload 103061568.tar to iLMS