Lab 3 **Modifying Sniper to Model Advanced Branch Prediction**
EE156/CS140 Advanced Computer Architecture
Prof. Mark Hempstead Due Date: 4/10/23
TA: TA: Parnian Mokri, Robert Costa and Bharat Kesari

1 Overview

This assignment will show you how to modify Sniper in order to test new hardware that hasn't already been implemented in the simulator. The assignment consists of the following parts:

- Implement the gshare branch predictor and integrate with sniper.
- Run an experiment and analyze the effects of different branch predictors.

2 Implementing GShare

In this part of the lab, you will implement the key functions required for the G-share branch predictor (described here: `http://www.hpl.hp.com/techreports/Compaq-DEC/WRL-TN-36.pdf`). Download the two shell files, gshare.h and gshare_test.cc, from Canvas. gshare.h is the gshare class and is where you will implement the **predict()** and **update()** functions. gshare_test.cc is a test harness that has been provided to you to help you verify the functionality of your implementation before integrating with sniper. The expected results of your gshare implementation is in the file expected_results.txt, which is also included on Canvas.

Note: You will need to modify `gshare_test` to output your results in the same format as `expected_results`.

You can compile and run using either g++ or gcc. For example:

$ g++ -Wall -Wextra -o gshare.o gshare_test.cc
$ ./gshare.o

Note: you will get the following warnings (and maybe more, depending on your compiler):

warning: unused parameter 'name' [-Wunused-parameter] GSharePredictor(char * name, int core_id, unsigned int pht_index_bits, unsigned int ghr_bits, unsigned int cntr_bits

warning: unused parameter 'core_id' [-Wunused-parameter] GSharePredictor(char * name, int core_id, unsigned int pht_index_bits, unsigned int ghr_bits, unsigned int cntr_bits

warning: unused parameter 'predicted' [-Wunused-parameter] void update(bool predicted, bool actual, int ip, int target)

warning: unused parameter 'target' [-Wunused-parameter] void update(bool predicted, bool actual, int ip, int target)

This is because the branch predictor functions in sniper are standardized to be compatible with a wide range of predictors, so all parameters may not always be used depending on the predictor.

3 Modifying sniper

Now that you have successfully implemented Gshare, it is time to integrate it with sniper.

3.1 Procedure

Note: Be careful if you copy/paste between this document and the terminal. Some characters don't copy as expected, like quotation marks and underscores.

1. Move gshare.h to /YOUR_PATH_TO_SNIPER/common/performance_model/branch_predictors/

2. Add the following "include" statement to the **top** of gshare.h: "#include "branch_predictor.h""
3. In gshare.h, change "class GSharePredictor" to "class GSahrePredictor : public BranchPredictor"
4. In gshare.h, change "GSharePredictor(char * name, int core_id, unsigned int pht_index_bits, unsigned int ghr_bits, unsigned int cntr_bits)" to "GSharePredictor(String name, core_id_t core_id, unsigned int pht_index_bits_, unsigned int ghr_bits_, unsigned int cntr_bits_):
BranchPredictor(name, core_id), pht_index_bits(pht_index_bits_) , ghr_bits(ghr_bits_) , cntr_bits(cntr_bits_)"
5. Remove the loop in the constructor of gshare.h
6. In the headers of the predict() and update() functions in gshare.h, change the types of "ip" and "target" from "int" to "IntPtr"
7. Uncomment the line in gshare.h which contains "updateCounters(predicted, actual);"
8. Change directories to /path/to/sniper/common/performance_model/
9. Open branch_predictor.cc
10. Add the following "include" statement to the top of branch_predictor.cc: "#include "gshare.h""
11. Add the following conditional statement to the conditional clause where it makes sense in the create() function of branch_predictor.cc:
"else if (type == "gshare")
UInt32 pht_index_bits = cfg->getIntArray("perf_model/branch_predictor/pht_index_bits", core_id);
UInt32 ghr_bits = cfg->getIntArray("perf_model/branch_predictor/ghr_bits", core_id);
UInt32 cntr_bits = cfg->getIntArray("perf_model/branch_predictor/cntr_bits", core_id);
return new GSharePredictor("branch_predictor", core_id, pht_index_bits, ghr_bits, cntr_bits);"
12. Change directories to /path/to/sniper/ and run make. You may need to point to a different version of gcc. To do this, run the following commands:
    (a) setenv PATH /usr/sup/gcc-7.4.0/bin:${PATH}
    (b) setenv LD_LIBRARY_PATH /usr/sup/gcc-7.4.0/lib64:/usr/sup/gcc-7.4.0/lib:${LD_LIBRARY_PATH}
    (c) setenv LD_RUN_PATH ${LD_LIBRARY_PATH}

Note: you will no longer be able to use the test harness.

4 Experiment

To explore the effects of using the GShare predictor, you will run an experiment testing different branch predictors as well as different configurations of the gshare predictor. Select 5 splash2 workloads which you have not already used for a previous lab. Use the gainestown architecture with 4 cores and a "small" input size. Test three different PHT sizes: 2^4, 2^8, and 2^{16}. Note that the number of entries in the PHT may not be an integer multiple of the index (hint: use modular arithmetic). Use a 32-bit GHR. Pick your favorite performance and power/energy metrics to use to analyze your results. You should compare with the default branch predictor for the gainestown architecture.

Note: To simulate the Gshare predictor you implemented and integrated with sniper, you will need to modify the .cfg file you are using. In this case, it is the gainestown architecture. If you open nehalem.cfg, you will see a section titled "[perf_model/branch_predictor]." This is where you will specify the type of predictor you will use as well as add its corresponding parameters. **Hint:** look at what you added to branch_predictor.cc in section 3.1. Here is an example (Notice that the numbers are arbitrary in this example):

```
[perf_model/branch_predictor]
type=gshare
mispredict_penalty=8
pht_index\_bits=2
ghr_bits=2
cntr_bits=2
```

5 Deliverables and Submission

You should upload four files to Trunk: gshare.h, gshare_test.cc, a text file containing your gshare test results (it should match those in expected_results.txt), and a report in PDF format documenting your experiment. **It is highly recommended that you use LaTeX for your report.** You should write your report from the perspective that you are a computer architect working in industry who is trying to inform his or her colleagues. Your report should include: introduction, experimental setup, results and analysis, and conclusion. Include topology, McPAT, and CPI diagram(s) when relevant. You should explain how varying the parameters affects your results (performance and power/energy). Then offer your analysis as to why you see the results you do. Do certain workloads perform better (in terms of both performance and power/energy) than others for certain predictors? Why do you think that is? Make sure to address the following questions wherever it is relevant in your report:

1. Describe at a high level the Splash2 workloads you chose to run.
2. Describe what it is that you are sweeping. For example, if you are sweeping prefetcher type, describe what a prefetcher is and describe the implementations of the prefetchers you are testing.
3. What performance metrics have you used to evaluate your architecture and why? What power/energy metrics have you used and why?
4. Analyze your results and make conclusions. Make sure you refer to your figures, and describe what you observe from the data. Don't leave it for the reader to interpret the data.

Note: You might want to use metrics that combine both power and performance, e.g., energy-delay-product (EDP); if this is the case, you need to state why you did that and how it helps you to evaluate your topology.

6 Provide

Submit the following files to provide:

- your gshare.h after you made the changes
- your updated branch_predictor.cc
- your gshare_test that uses updated gshare.h
- your Makefile, or instructions on how to make your updated gshare_test
- The output of updated gshare_test in a text file. **TA's note:** We will use a diff between your final result and the expected result and grade you based on the diff result. So, try to follow the same flow.
- Your report as a PDF file.Make sure you refer to plots in the text of your report, point to the neat observation in each figure in the text and explain why it is happening.

To Provide:

- ssh to homework or any other server : ssh homework
- provide ee156 lab3 FILENAME.zip

7 Tips and tricks

- Error on Make: Check the path to GCC. See point 12, section 3.1
- You will need to modify `gshare_test` to output its results in the same format as `expected_results.txt`
- Focus on the single pattern global history table in the paper, this is what you need to implement.