CS425: Computer Systems Architecture

Programming Assignment 1
Assignment Date: 03/12/2021
Due Date: 17/12/2021 – 23:59

Instructions: Put your answers in a .pdf file and send them together with your source code via e-mail to HY425 course e-mail (hy425@csd.uoc.gr). Set the e-mail subject: HY425 – Programming Assignment 1.

Branch Prediction

The purpose of this assignment is to introduce you in simulation and help you familiarize with the details of branch predictors. You have to implement branch predictors yourself and measure their quantitative properties. You will also see the impact of alternative design choices in the accuracy of predictions.

For the simulation of branch predictors you will use the gem5 simulator (www.gem5.org/). Although the assignment does not require prior experience with gem5, it is recommended that you take some time to follow the gem5 tutorial:

www.gem5.org/documentation/learning_gem5/introduction/

Simulator

In this assignment you will compile gem5 to target the x86 ISA. The build requirements can be found here: www.gem5.org/documentation/learning_gem5/part1/building/
You are able to work on this assignment solely on your own environment. There is no need to use the department's workstations but you are free to do so, as well.
You can get a snapshot of the simulator by cloning this repository:

https://github.com/sotot0/cs425_PA1.git

The repository contains:

i. the gem5 directory,
ii. and the benchmark directory

After you have installed the indicated dependencies, clone the repository and compile the simulator by typing:

```
python3 `which scons` build/X86/gem5.opt -j9
```

inside the gem5 directory. The compilation step consumes time so complete it as soon as possible!

To run a batch of five (5) benchmarks with the simulator you have to use the following command inside the benchmark directory:

```
./runall.sh O3CPU StaticPred 256
```

This command will run all the benchmarks while simulating a setting with an Out-Of-Order CPU (O3CPU), a static branch predictor (static-taken) supported by a Branch Target Buffer (BTB) of 256 entries. You are encouraged to examine thoroughly the runall.sh / README files in order to figure out the different arguments that you are able and allowed to pass. The arguments that are related to the branch predictors (StaticPred, LocalBP, GapPred, PAgPred) can be changed easily from the

command line. However, arguments that are related to the cache and CPU domain will remain as they are during this assignment. Also, note that the -I 50000000 argument means that for each benchmark gem5 simulates 50 million instructions. You can change this parameter during development but for generating final numbers you should have at-least 50 million instructions. If you set a very large number of instructions, then simulation time will increase dramatically (even with the parameter set to 50 million instructions you may experience long simulation times especially when running benchmark1/benchmark4). Ensure that your comparisons are based on the same instruction count between different benchmarks and/or settings to extract uniform and fair results.

Implement Branch Predictors

Your main task is to implement in the simulator two branch predictors: (1) **GAp** and (2) **PAg** and measure their performance using the given benchmarks. Study the slides presented in the class and the paper from Yeh and Patt: "Alternative implementations of two-level adaptive branch predictors" for further details - you can find it in the website of the course.

It is recommended that you implement the predictors in a parametric fashion, (check the 2bit_local.hh, 2bit_local.cc and the BranchPredictor.py in */src/cpu/pred*). The infrastructure to feed the already implemented branch predictors and your GAp or PAg with parameters is already implemented. You only have to implement them in GApPredictor.hh/.cc and in PAgPredictor.hh/.cc taking into account the corresponding python classes in BranchPrediction.py. In this way you will be able to run easily different configurations of predictors (e.g. number of table entries, history, predictor sizes etc.). You are encouraged to examine carefully the bpred_unit.hh/.cc files because your assigned branch predictors will inherit from this base class. Also, member functions of this class implement the book keeping of various statistical information. Do not forget to put comments in your code!

After you finish with the implementation (and debugging!), pick a number of different predictor configurations (specify your choices) for GAp, PAg, Static and Local, run all benchmarks and provide graphs and/or tables showing the results in terms of IPC and Branch Misprediction rate. Write on your report your findings and pair them with your explanations. Also, recommend a specific setting that you observed it provides the best possible statistics when simulating it. It is important to get familiar with the generated m5out directories under /benchmarkN directories after each run. These directories include files showing the simulated configuration and its statistical information.

O3CPU	***Static (choices)***		***Local (choices)***		***GAp (choices)***		***PAg (choices)***	
	IPC	BMR	IPC	BMR	IPC	BMR	IPC	BMR
benchmark1								
benchmark2								
benchmark3								
benchmark4								
benchmark5								

Now, let's assume that you have a tight memory budget for the predictor tables, expressed in total number of table entries. Which branch predictor and with what configuration would you choose if you had a budget of 2048 entries (both for the predictor and BTB)? Run experiments, explore your choices and calculate the prediction rates! Report your findings using tables and/or graphs.

The files you must deliver are the following:

- GApPredictor.hh and GApPredictor.cc
- PAgPredictor.hh and PAgPredictor.cc
- Any custom scripts that you used
- Your report in PDF format
- m5out directories that contain statistics along with the corresponding configurations

All codes will be analyzed for copying using special software!