

CS425: Computer Systems Architecture

Simulation Assignment 1

Assignment Date: 15/11/2024

Due Date: 02/12/2024 - 23:59

Instructions: Put your answers in a .pdf file and send them together with your source code via email to HY425 course e-mail (hy425@csd.uoc.gr). Set the email subject: HY425 – Simulation 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 in 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 typing the following git command:

```
git clone -b hy425_sim1 https://github.com/sotot0/gem5.git
```

This command will automatically clone from the main repository and checkout to hy425_sim1 branch of it. You will work only on this branch.

The repository contains:

1. the gem5 main directory structure, and
2. the hy425 directory which includes benchmarks

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

```
python3 `which scons` build/X86/gem5.opt -j <Avail. Cores>
```

inside the main directory. The compilation step takes 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 hy425 directory:

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

This command will run five (5) benchmarks while simulating a setting with an Out-of-Order CPU, a static branch predictor (static, not-taken) supported by a Branch Target Buffer (BTB) of 128 entries. You are encouraged to examine thoroughly the runall.sh and README files under benchmark directory 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, GAgPred, PAgPred) can be changed easily from the command line. Due to long simulation times while simulating the Out-of-Order CPU, you have the option of changing the CPU_TYPE to TimingSimpleCPU. TimingSimpleCPU is a simpler thus much faster CPU model (in terms of simulation time). But you are allowed to do so only while debugging your implementations and for testing purposes. The deliverable statistics and final results have to be measured on the OoO type.

However, arguments that are related to cache domain will remain as they are during this assignment. Also, note that the -I 50000000 argument indicates 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 an even larger 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 benchmarks 1 and 4). 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) **GAg** 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 on the website of the course.

It is recommended that you implement the predictors in a parametric fashion, (check 2bit_local.hh, 2bit_local.cc and BranchPredictor.py in /src/cpu/pred). The infrastructure to feed the already implemented branch predictors and your GAg or PAg with parameters is already implemented. You only have to implement them in GAgPredictor.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 easily run different configurations of predictors (e.g. number of table entries, history, predictor sizes etc.). You are also encouraged to examine carefully the bpred_unit.hh/.cc files because your assigned branch predictors will inherit from this base class. Member functions of this class

implement the bookkeeping of various statistical information. Do not forget to put explanatory comments in your code!

After you finish with the implementation (and debugging!), pick a number of different predictor configurations (specify your choices) for GAg, PAg, Static and Local, run all benchmarks and provide graphs and/or tables showing the results in terms of IPC and Branch Misprediction Rate (BMR). Write on your report your findings and explain them. Recommend a specific setting that you observed that 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.

<i>O3CPU</i>	<i>Static (choices)</i>		<i>Local (choices)</i>		<i>GAg (choices)</i>		<i>PAg (choices)</i>	
	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 the total number of bits. Which branch predictor and with what configuration would you choose if you had a maximum budget of 8192 bits for both predictor levels? (Do not count the first level of GAg which is just a register). Run experiments, explore your choices and calculate the prediction rates! Report your findings using tables and/or graphs.

The files you must deliver the following items (in a compressed tarball e.g. zip, gz, bz2):

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

All codes will be analyzed for copying using special software!