HY425: Computer Systems Architecture

Programming Assignment 1
Assignment: 28/10/2016
Due: 13/11/2016 23:59

Instructions: Solve all problems in a .pdf file and send them via e-mail to Dimitris Giannopoulos (dgiannop@csd.uoc.gr). **Use the subject: HY425 – Programming Assignment 1**

Branch Prediction

The purpose of this assignment is to introduce you in simulation and familiarize yourself 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 our custom simulator that is based on the PIN dynamic binary instrumentation tool (www.pintool.org). Although the assignment does not require prior experience with PIN, however, it is recommended that you take some time to follow the PIN tutorial:

http://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool

Simulator

The simulator is designed to work only on x86 machines (PIN requirement) and the recommended OS is Linux with gcc >= 3.4 and Bash shell – the Linux workstations of the department are sufficient. You can get the simulator from the home directory of the course:

/home/misc/courses/hy425/HW/2013f/PA/PA1_BranchPrediction

The directory contains:

(i) the instrumentation tool (BranchPrediction.cpp),
(ii) two demo branch predictors that you can use as reference (StaticPredictor.H, NbCounterPredictor.H),
(iii) a Makefile and (iv) a directory with 7 benchmarks (4 integer and 3 floating point) to exercise the branch predictors and measure their performance.

Copy the simulator in your home directory and study carefully the sources to familiarize with the simulator. There are plenty of comments to guide you on how to implement branch predictors. To compile the simulator just type: ***make***.
Notice that the Makefile references the PIN installation from the home directory of the course, which is sufficient if you use the simulator from the workstations of the department. If you want to run it from your own PC, then you will have to install PIN (2.13) and modify a few header lines in the simulator's Makefile. Keep in mind that we cannot support the numerous Linux distributions/versions etc, so we do not promise to solve any issues that might arise in your PC.

To run a benchmark with the simulator (e.g. fft) use the following command:

make SIM_ARGS="-bht 12" SIM_APP="./benchmarks/6.fft/fft" run

SIM_ARGS sets the command line arguments that you can pass to the simulator (check the BranchPrediction.cpp KNOBs to see the available switches) and SIM_APP sets the application/benchmark that will run on the simulator. Note: simulating each of the given

benchmarks takes a couple of minutes to complete (they contain a few hundred million branches) ! If you want to simulate something small, then you can write a dummy hello world program in C, compile it as usual, and feed the executable binary to the simulator by setting SIM_APP appropriately. (You can even simulate relatively large applications, e.g. firefox)

Implement Branch Predictors

Your main task is to implement in the simulator two branch predictors: (1) GAp and (2) PAp 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 NbCounterPredictor.H and the KNOBs of BranchPrediction.cpp) so that you can run easily different configurations of predictors (e.g. number of table entries, associativity etc). Moreover, implement the appropriate statistics collection and try to enrich them with useful metrics. Do not forget to put comments in your code!

After you finish with the implementation (and debugging!), pick one GAp configuration and one PAp configuration (specify your choices), run all benchmarks and fill the following table with the observed prediction rates (fill also the rates for the demo predictors using their default configurations). Do not forget to do warmup (10K-100K is fine)!

	Predictors			
Benchmark	***Static***	***NbCounter***	***GAg (choice)***	***PAg (choice)***
1. bzip2 (INT)				
2. bunzip2 (INT)				
3. mcf (INT)				
4. qsort (INT)				
5. matmul (FP)				
6. fft (FP)				
7. fish (FP)				

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

We analyze all codes with MOSS!