ECEN/CSCI 5593 Advanced Computer Architecture (ACA)
Checkpoint 1A: Branch Prediction Simulation with PIN

Overview:

Use the PIN tool to instrument applications to implement a small simulator to evaluate the prediction performance (accuracy) of a branch-prediction buffer with various branch prediction schemes.

Single-bit predictor: a 1-bit predictor storing (taken or not-taken last time)
Bi-model bit predictor: a 2-bit predictor saturating counter
Two-level (GAg) predictor using a 8-bit history register and a 2^8 entry pattern history table. Each pattern table entry has a 2-bit saturating counter.
Two-level (PAg) predictor using a 12-bit local branch history and a 2^12 entry pattern history table. Each pattern table entry has a 2-bit saturating counter.

For the 1-bit and 2-bit schemes, entries in the BTB tables are only initially created when a branch is first taken (non-taking branches will by default be predicted as non-taken since they have no previously created table entry). Global history for the GAg scheme is always updated whether the branch is in the BTB or if it is an always-taken branch.

Work:
Construct and model a direct-mapped Branch Target Buffer (BTB) table in C/C++ programming language to simulate the branch prediction mechanism in a processor. The BTB should be a programmable number of entries, assume there are 1024 entries for this assignment. Each entry contains: prediction information (either 1-bit, 2-bit, or 12-bit local history), and tag information for verifying the correct branch address in the BTB entry. The BTB entry holds the branch information for prediction. In the case of the GAg predictor, the 8-bit history register is shared by all of the branches executing. Since there are 1024-entries, a 10-bit field will be used from the executing branch IP to locate its BTB entry. The 10-bit field will start at the lowest bit.

Assume that the two-bit counters in the pattern history table are all initialized to weak-taken (10) when a branch is first entered.

Assume that the system "clear (or reset) a branch entry's local history register if a new branch replaces an existing branch (map to same BTB entry, but not the same tags).

The GAg and PAg systems should also start the history registers of all not-taken as the starting history (000000000).

Tasks:

1- Determine the prediction accuracy of each predictor for FOUR of the program applications using PIN's ability to run : 400.perlbench 401.bzip2 403.gcc 429.mcf 445.gobmk 456.hmmer 458.sjeng 462.libquantum 464.h264ref 471.omnetpp 473.astar

You are only to run the programs for 10,000,000,000 branch instructions, and then you should end the program. You will want to build a PIN Knob which allows you to control this setting, so that you can start by using only 10,000 or so (a short simulation time).

2-Develop a new unique predictor, explain the predictor, and report and the predictor's accuracy. There could be a number of new branch predictor schemes. You have to create on your own idea, you do not have to do one of the PAg, SAg, SAs, etc, types. You can invent your own (hybrid, etc). Alternatively, instead of building a new branch predictor and simulating it, you can evaluate interesting statistics about the branch predictors: streaks of correct/incorrect predictions, analysis of really bad branches, etc.

3- Answer the following by performing experiments.
What is the prediction accuracy of each scheme, for each application?
What is the miss rate (not finding a branch in the BTB) for each application?

Does using a longer global history for the GAg scheme, provide better prediction accuracy for the different application benchmarks?

Which applications have the most global correlation and why?

[Extra Credit] How many branches are executed for each application, how much do the top branches account for in execution (top 5), how many of the mispredicts are accounted by the top 5 executing branches.

Assignment:
Upload .tar file with .pdf file report with graphics and analysis and your code.
Example: >> tar -cvf connors.tar report.pdf *.C *.h

Getting Started:

copy files from (on your eces-shell account/machine):
/cad-scratch/5593/checkpoint1A

These are the files for getting started.

Also copy the following directory to your home directory (and there is a README file)
/cad-scratch/5593/benchmarks
cp -r /cad-scratch/5593/benchmarks .

Checkout the $HOME/benchmarks/README which explains how to run the
real applications through your branch predictor.

Currently, checkpoint1.C is configured to add instrumentation to count all of the branches in the program. All you have to do is modify it to call a function that keeps track of predictions.

To generate a shell script that runs a pin tool on one of the spec2006 integer applications (running on the training input set)

bench-run.pl --bench spec-cpu2006:int:401.bzip2:train --build base --prefix "pin -t <pin_tool_name> --" --copy output.out --log LOG

this will create a script go.sh which will you run at the command line as:

./go.sh

While go.sh is running, you can peek into the LOG file to see the progress.

You may want to run "./go.sh &" and go for lunch while it is computing the simulations. But be ready to analyze the data when you return.

output.out will be copy after each application to the directory of
that application: src/spec_cpu2006/int/401.bzip2/builds/base/output.out