CSE 240A Project 1 – Branch Predictor Contest

Due October 29th 11:59PM

This project is based on the 2004 Championship Branch Predictor contest. Each contestant will write a branch predictor that will consume a trace of branches (generated from real execution). It will be based on a standard interface. The predictor will have a budget, namely (32K + 256) bits (not bytes). That encompasses all storage (all tables, plus GHR, if necessary -- but not PC) used for the predictor. That may not be the amount of storage your predictor simulator uses, however -- for example, you may implement 2-bit predictors with a table of ints, in which case the simulator will use more memory -- that's okay, we're only concerned about the memory used by the simulated branch predictor. The output of your simulator will show number of branches, number of mispredicted branches, and mispredict rate (shown as mispredicts per 1K instructions). Your grade will depend almost exclusively on the last item, however.

You will be provided with a complete interface to the traces, etc., significantly minimizing the amount of work you have to do. So even if you don't like C/C++, it will be easier to use this than to write your own predictor from scratch. You'll simply provide three functions, as described in the README file.

Like the original contest, we will use two sets of traces -- one to allow you to test the predictor, and another (hidden from you) to evaluate. The test traces will be available to you. We will run your predictor on the final traces. You will turn in a full report, including your final code, by Oct 29 11:59PM. We will determine the winners by which predictors run best on the final traces.

Points for the project will go something like this. 50, 49, 48, 47, 46, 45, 44, 43, 42, 41 points for 1st through 10th place. 40 points for any predictor above the high standard, and 35 for any predictor above the low standard. Lower points for predictors below standard. Much lower for buggy predictors. The standard is established like this: We have two reference predictors, an Alpha 21264-like predictor (within the size constraints), and a 2-level local predictor. The better of those predictors, overall, will represent the high standard you have to beat, and the other will represent the low standard. Keep in mind that beating these predictors for the test traces does not guarantee success on the final traces, so make sure you are well ahead. And we'll do some tuning of these predictors within the size constraints to get good performance.

Rules.

1. Write your own code. Don't look at code from others, in class or outside of classes, or from a previous class, or off the web. There are all kinds of places to get branch predictor code, if you wanted to. Don't.

2. Do not reference the predictors from the contest (either the code or the descriptions). However, there are a whole set of predictors cited below that should prove helpful. There are plenty more if you keep looking (including more recent proposals). Some of the contestants later wrote papers about their predictors -- if you stumble on one of those, that's fine.

3. Use C or C++, ideally on CSE machines. However, as long as you can get access to a UNIX account, and use our framework, you should be okay. Just keep in mind that we need to be able to run your code without problems.

4. Turn in the code and a short report, and mail them to Rajib (see turn-in instructions below). The report has four parts: (a) name of your branch predictor, (b) a one sentence description of your predictor, (c) a table with all your results, including arithmetic means over all traces, (d) a complete description of the predictor. Include citations of any predictor you are based on.

5. This is an individual project. You may want to work in groups to survey the available literature, but split up before you decide on your predictor so we get more interesting variety. Turn-in Instructions.

1. Create a directory with your eng.ucsd.edu name. For example, if your email id is xyz@eng.ucsd.edu, your directory name would be xyz.

2. Copy predictor.C, predictor.h, and report.pdf to the directory.

3. Make a tar ball using tar –cvzf <dirname.tgz> <dirname>

4. Send an email to Rajib at rknath@cs.ucsd.edu with the tar ball as an attachment. The subject of the email should be CSE240A Project 1.

Any submission that does not follow the above guidelines will not be graded.

Hints. 1. Code the predictors you need to beat -- don't guess. 2. Make sure you beat them by a good margin.

You can download all the files from Piazza.

References.

These are all a bit aged. I'll leave finding some of the more recent developments up to you.

McFarling, Combining Branch Predictors, WRL TN-36. Good description of local, correlating, gshare, and combining predictors.

Kessler, The Alpha 21264 Microprocessor, IEEE Micro, 1999. Uses variant of McFarling's combining (tournament) predictor.

A. Eden, and T. Mudge. The YAGS branch predictor. 31th Ann. IEEE/ACM Symp. Microarchitecture (MICRO-31), Dec. 1998. Good description of a variety of anti-aliasing predictors (predictors that still work with large working sets of branches). Don't take their word that theirs is best without testing it out...

C.-C. Lee, I.-C. Chen, and T. Mudge. The bi-mode branch predictor. 30th Ann. IEEE/ACM Symp. Microarchitecture (MICRO-30), Dec. 1997. Although the previous had a good description of bi-mode, this is the original.

A. Seznec, S. Felix, V. Krishnan, Y. Sazeides. "Design trade-offs for the Alpha EV8 conditional branch predictor", in : Proceedings of the 29th International Symposium on Computer Architecture, May, 2002. Take a deep breath before you enter...

D. Jimenez, C. Lin, "Neural Methods for Dynamic Branch Prediction" in ACM TOCS, November 2002. Uses ideas from neural networks (perceptrons) to implement a branch predictor.