CS 6290: High-Performance Computer Architecture

Fall 2015

Project 1
Due: September 23rd 2015 at 5pm EST

This project is intended to help you understand branch prediction and performance of out-of-order processors. As with Project 0, you will need the "CS6290 Project VM" virtual machine for this project.

You will fill in the form fields in this PDF document and submit it in T-Square (the submitted file name should still be PRJ1.pdf). Note that you will need to save the filled-out form, and that many PDF readers/viewers cannot save the document. One free online viewer that you can use to do this is http://www.pdfescape.com/. If you are having trouble filling (or saving) this PDF form, fill out and submit the provided PRJ1.txt form. If you do use the PRJ1.txt file, please note that the answers you put into PRJ1.txt should be **exactly** the same as what you would enter in the PDF form, and that the format of the PRJ1.txt file must be preserved

Additional files to upload are specified in each part of this document. Do **not** archive (zip, rar, or anything else) the files when you submit them – each file should be uploaded separately, with the file name specified in this assignment. You will lose up to 10 points for not following the file submission and naming guidelines. Furthermore, if it is not VERY clear which submitted file matches which requested file, we will treat the submission as missing that file. The same is true if you submit multiple files that appear to match the same requested file (e.g. several files with the same name). In short, if there is any ambiguity about which submitted file(s) should be used for grading, the grading will be done as if those ambiguous files were not submitted at all.

As explained in the course rules, **this is an individual project**: **no collaboration with other students or anyone else is allowed on this project**.

Part 1 [20 points]: Configuration of the Branch Predictor

The hardware of the simulated machine is described in the configuration file. In this project we will be using the cmp4-noc.conf configuration file again, but this time we will modify this file so this is a good time to make a copy so we can restore the original configuration when we need it.

The processors (cores) are specified in the "cpucore" parameter near the beginning of the file. In this case, the file specifies that the machine has 4 identical cores numbered 0 through 3 (the procsPerNode parameter is 4), and that each core is described in section [issueX]. Going to section [issueX], we see that a core has a lot of parameters, among which we see that the clock frequency is set at 1GHz, that this is an out-of-order core (inorder set to false) which fetches, issues, and retires up to 2 instructions per cycle (the "issue" parameter is set to two earlier in the file). The core has a branch predictor described in the [BPredIssueX] section, fetches instructions from a structure called "IL1" described in the [IMemory] section (this is specified by the instrSource parameter, and reads/writes data from a structure called "DL1" described in the [DMemory] section. In this part of this project, we will be modifying the branch predictor, so let's take a closer look at the [BPRedIssueX] section. It says that the type of the predictor is "Hybrid" (which does not tell us much). This is actually a tournament predictor, with a meta-predictor that has 2048 entries, each of which is a saturating counter.

You need to figure out what are the two component predictors for this tournament predictor. The source code for all of the branch predictors is in BPred.h and BPRed.cpp in the ~/sesc/src/libcore/ directory. Hint: The "Hybrid" predictor type is simulated using the BPHybrid class, so you should look at the code for BPHybrid::predict to see what this predictor actually does, then fill in the blanks in the following statements:

A) The predictor that is using l2size and l2bits parameters is a N-Bit Predictor (Counters C predictor with a (enter N/A here if this predictor is not a history-based one) ________-bit history. The PC of the branch instruction and this history (if one is used) are combined to index into a ________-entry counter array, where each entry is a ______-bit counter.

B) The predictor that is using localsize and localBits parameters is a N-Bit Predictor (Counters C predictor with a (enter N/A here if this predictor is not a history-based one) ________-bit history. The PC of the branch instruction and this history (if one is used) are combined to index into a ________-entry counter array, where each entry is a ______-bit counter.

Part 2 [30 points]: Changing the Branch Predictor

Now we will compare some branch predictors. The LU benchmark we used in Project 0 does not really stress the branch predictor, so we will use the raytrace benchmark:

```
cd ~/sesc/apps/Splash2/raytrace
```

```
make
```

No we will simulate the execution of this benchmark using the unmodified cmp4-noc configuration (with the "Hybrid" predictor).

```
~/sesc/sesc.opt  -c  ~/sesc/confs/cmp4-noc.conf  -ort.out
 -ert.err raytrace.mipseb -p1 -m128 -m128 Input/reduced.env
```

Now we will modify the configuration file, so make a copy of it if you did not do this already. Then change the configuration to model an oracle (perfect) direction predictor by changing the "type" of the predictor from "Hybrid" to "Oracle", then and re-run the simulation. Note that overall branch prediction accuracy is not perfect in this case – only the direction predictor is perfect, but the target address predictor is a (non-oracle) BTB!

After that, configure the processor to use a simple predict-not-taken predictor (type="NotTaken") and run the simulation again.

C) Submit the simulation report files for these three runs (rename them to **sesc_raytrace.mipseb.NTC, sesc_raytrace.mipseb.HyC, sesc_raytrace.mipseb.OrC**) along with your answer form in T-square. Note that these are the **simulation-generated reports**, not what the report.pl script produces. You will lose 5 points in Part 2 for each missing report file.

D) In the table below, for each simulation fill in the overall accuracy (number under BPred in the output of report.pl), the number of cycles, and the speedup relative to the configuration that uses the simple NotTaken predictor.

	BPred Accuracy	**Cycles**	**Speedup vs. NotTaken**
NotTaken	%		
Hybrid	%		
Oracle	%		

E) We get a large speedup when we switch from the NotTaken to the Hybrid predictor, but the accuracy improves by an even bigger factor (Hybrid accuracy divided by NotTaken accuracy). Explain why the speedup does not improve as much:

F) Now change the processor's issue parameter to 4 instead of 2 and repeat the three simulations (Hybrid, Oracle, NotTaken). Submit the simulation report files for these three runs (rename to **sesc_raytrace.mipseb.NTE, sesc_raytrace.mipseb.HyE, sesc_raytrace.mipseb.OrE**). Remember that you will lose 5 points in Part 2 for each missing report file!

G) In the table below, fill in the IPC achieved with each type of predictor when the processor is 2-issue and when it is 4-issue, then compute the speedup of going from 2-issue to 4-issue for that type of predictor.

	Cycles w/ 2-issue	**Cycles w/ 4-issue**	**Speedup of going from 2- to 4-issue**
NotTaken			
Hybrid			
Oracle			

H) The results in E) lead us to conclude that improvements in branch prediction become irrelevant as the processor core becomes capable of executing more instructions per cycle. Explain why:

Part 3 [50 points]: Implementing a New Branch Predictor

In this part of the project we again use the cmp4-noc configuration. **You should change it back to its original content, i.e. what it had before we modified it for Part 2**. We will continue to use the Raytrace benchmark with the same parameters as in Part 2.

Our goal in this part of the project will be to create a hierarchical predictor that achieves similar accuracy but is less expensive than the Hybrid one that already exists in the simulator. The new predictor we will a hierarchical one. You should start out by making a duplicate of the BPHybrid class in Bpred.h and BPred.cpp. The new class should be named BPHier, and the "Hybrid" string in the predictor's constructor (search for "Hybrid" in BPred.cpp, including the quotes in the search string) should be changed to "Hier". You should change the BPredictor::getBPred method (in BPred.cpp) to create a BPHier when the predictor type in the configuration file is "Hier" (see how BPHybrid is created there).

Now you need to change the code of the predictor to become a hierarchical instead of a tournament predictor – and remember to recompile the simulator after you change its source code! We will no longer need the meta-predictor (so remove the "metaTable" from your BPHier class). Instead, on each prediction we will first consult (and update, if needed) the localTable predictor. If the counter in localTable is fully saturated[1] we will use that as the final prediction, and the more sophisticated ("globalTable") sub-predictor will neither be consulted nor updated. If the prediction of the localTable is not fully saturated, we will consult (and update, if needed) the globalTable predictor and use its decision as the final one. Note that the global history register needs to be updated on every conditional-branch outcome, even if the history is not needed for that prediction.

Recall that the goal of the hierarchical predictor is to avoid using space in the more sophisticated predictor if a less-sophisticated predictor can provide a good prediction for that branch. In this implementation we are using counter saturation as a proxy for getting good prediction from the localTable predictor. The hierarchical predictor described above is an attempt to do that, and you should implement the predictor as described above.

[1] Strongest possible "Taken" or strongest possible "Not Taken" state, you can use isLowest and isHighest methods in the SCTable class (which models the table of counters) for this. See ~/sesc/src/libsuc/SCTable.h

I) When I use the original configuration file and only change the predictor type from "Hybrid" to "Hier", the Raytrace application (same simulator command line as for Part 1) gets the predictor accuracy of ________% and Raytrace executes in ________ cycles, which represents a speedup of ________ relative to the default Hybrid predictor.

J) Submit the simulator's report file (rename it to **sesc_raytrace.mipseb.HiJ**) for the simulation using your new predictor. You will lose 10 points is this file is missing. Also **submit your modified BPred.h and BPred.cpp** source code files. You will lose all points in Part 3 if these source code files are missing. You should not modify any other files in the simulator's source code.

Now we will try to reduce the size of the globalTable predictor. Reduce the l2size parameter from 2*1024 to only 256 for the Hybrid and for our Hier predictor.

K) Submit the simulator's report files (rename them to **sesc_raytrace.mipseb.HiK** and **sesc_raytrace.mispeb.HyK**)

L) What is the predictor accuracy, execution cycles for each of these configurations, and what is the speedup of each relative to the same-predictor run that had l2size=2*1024?

	BPred Accuracy for l2size=256	**Cycles for l2size=256**	**Speedup vs. l2size=2*1024**
Hybrid	%		
Hier	%		

M) The hierarchical predictor avoids use of the globalTable for branches that are predicted well in the localTable. This is supposed to allow the globalTable size to be reduced without affecting performance much. Which predictor is affected less by this large reduction in l2size? The existing hybrid predictor

N) The Hier predictor that uses l2size=2*1024 is smaller than the Hybrid predictor that has l2size=256 (Hybrid has 2048 more bits in the metaTable and 1792 fewer bits in the globalTable). Which one performs better? Hier with l2size=2*1024

O) We expected Hier to be similar to Hybrid when given equal table sizes, and to be better than Hybrid when we reduce the globalTable size. The actual results are different from our expectations. What do you think is the reason?