CSE 520 Computer Architecture -- Spring 2019

Programming Assignment 3 (100 points)

In this assignment, you will implement gDAC branch predictor [1] and compare its performance with other built-in branch predictors in gem5. In addition, you need to provide a way to specify branch predictor configuration in the command line (similar to what you did on replacement policy in Assignment 2). The implementation should be based on a latest gem5 commit. It should contain modifications to the existing gem5 source code and additions of new source files. You can git clone this commit from the link (This is the same commit you used in Assignment 2):

https://github.com/gem5/gem5/tree/34b73dea1b144fcc5707d618acd950f7f1506806

Branch Prediction in Gem5

There are several branch predictors implemented in gem5, including 2-bit local predictor, bi-modal predictor, and tournament predictor. However, gem5 does not expose any branch prediction configuration in the command-line interface. src/cpu/o3/O3CPU.py is the file where the particular branch prediction is selected, whereas src/cpu/pred/BranchPredictor.py specifies the configuration of each predictor. All the branch prediction implementation can be found in src/cpu/pred as well.

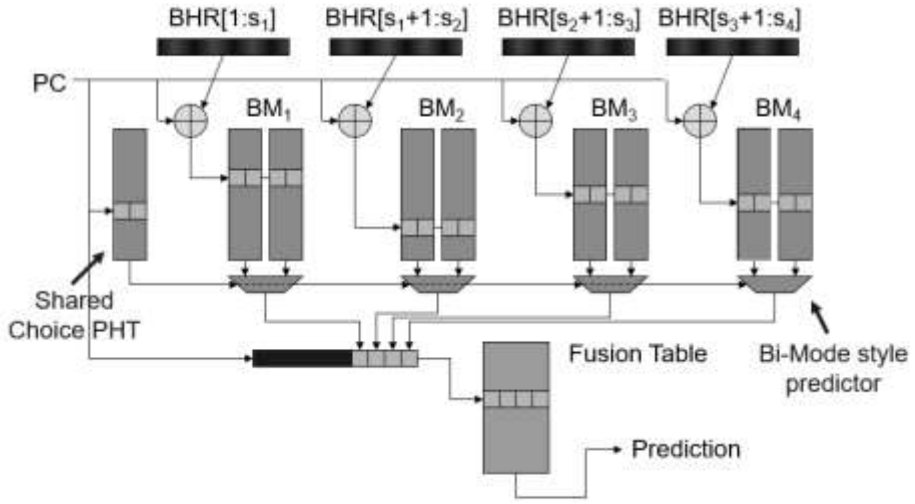
A branch predictor in gem5 first inherit the base class, BPredUnit, from src/cpu/pred/bpred_unit.hh. The different behaviors of predictors are distinguished in the 5 **virtual** functions, **lookup**, **update**, **uncondBranch**, **btbUpdate**, and **squash**. Here are the descriptions and notes of each virtual function.

- **lookup**
  This function returns whether a branch with a given PC is predicted as taken or not taken. The counter data of branch history (Like the counter to store the global history and the counter for local history for the specific PC) are also backup in this stage, and can be used to restore those counter if necessary. In some predictors, you might wonder why they also update global/local history in this stage with predicted value while the true branch outcome is under calculated. This action is primarily for the following branch instructions (if any) after the current one. Those branch instructions need to know the history, including what have been predicted. That is also why you will see the following functions try to correct the history if they were not as expected.
- **update**
  Update those counter value of history in any mean for branch prediction by a given outcome of a branch instruction. Note that the function argument, **squashed,** is to indicate whether this is a misprediction or not.
- **uncondBranch**
  This function is called when the target branch instruction is an unconditional branch. Typically, a branch predictor will do nothing but update global history with taken, as a unconditional branch is always taken.
- **btbUpdate**
  This function is called only if there is a miss in Branch Target Buffer (BTB). It means the branch prediction does not know where to jump even though the predictor can accurately predict the branch outcome. In this case, you will predict a branch as not taken so that the target branch address is not required. Thus, this function is typically to correct what you did in the lookup function into a result that you predict a branch as non-taken.
- **squash**
  If there is a branch misprediction, everything changed (counters, registers, cache, memory, etc) due to the outstanding instructions after this branch need to be reverted back into the state before the

branch instruction was issued. This includes those victim branch instructions following the branch misprediction. This function is primarily called to erase what the outstanding branches did to those history counters by restoring them with the backup value you stored in **update** function.

The branch predictors needed to be implemented and compared in this assignment

- 2-bit local (built-in)
- Tournament (built-in)
- gDAC (your implementation): The organization of gDAC branch predictor [1] is shown in the following figured (extracted from the presentation slide in PACT 2005 [3]). For the fusion table, please refer to [2].



We would like you to play a bit more with different configurations for the predictors. Following table shows the 2 groups of configuration for branch predictors. To make a fair comparison across predictors, the estimated total storage size of 3 branch predictors in each group will be roughly the same. Then, you can observe how size can affect the performance of one predictor by the configurations across groups

Storage size	2-bit local	Tournament	gDAC
Group 1 (32Kb)	16384 entries * 2-bit	● Local history table - 1024 entries * 11 bits ● Local predictor size - 2048 entries * 2 bits ● 1x 12-bit Global history register ● Global predictor size - 4096 * 2 bits ● Choice predictor size - 4096 * 2 bits (31744 bits in total)	● Use the configuration in Table 1 of [1]
Group 2 (8Kb)	4096 entries * 2-bit	● Local history table - 256 entries * 9 bits ● Local predictor size - 512 entries * 2 bits ● 1x 10-bit Global history register ● Global predictor size - 1024 * 2 bits ● Choice predictor size - 1024 * 2 bits (7424 bits in total)	● Use the configuration in Table 1 of [1]

In gem5 command, you only enable the following flags and leave all others as the default

- --cpu-type=DerivO3CPU
- --caches
- --l2cache

- --bpred= TournamentBP(), LocalBP(), and gDACBP()

Similar to what you hand in during the assignment 2, you have to give a brief introduction to the 3 branch predictors in the report, followed by the table and graph showing the **IPC** and **branch prediction accuracy** running the predictor configuration against the 3 compiler-optimized workloads (as follows), then draw your observation and conclusion.

- BFS with input R10K.graph
- MST with input rand-weighted-small.graph
- queens with option "-c 11"

[1] Gabriel H. Loh. A Simple Divide-and-Conquer Approach for Neural-Class Branch Prediction. In Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques, St. Louis, MO, USA, September 2005.

[2] O. H. Loh and D. S. Henry. Predicting Conditional Branches with Fusion-Based Hybrid Predictors. In Proceedings of the 11th International Conference on Parallel Architectures and Compilation Techniques, Charlottesville, VA, USA, September 2002.

[3] http://pact05.ce.ucsc.edu/neuralbp.ppt

Due Date

The assignment is due by April 2 at 11:59pm.

What to Turn in for Grading

1. Create a working directory, named "cse520-assgn03-LastName_FirstInitial", for the assignment to include
   a. A pdf report to include a description of your implementation, and performance data in tables, and/or graphs. Don't forget to add your name and ASU id in the report.
   b. A patch to include all your changes in the specific gem5 commit for the implementation of gDAC branch predictor.
   c. A readme text file with all the commands you use. Alternatively, you could have a script file with comments inside on how to use it.
   d. gem5 output file "stat.txt" of the gem5 execution runs. The name of the files should be named as "benchmark_branchPred _Group_stat.txt", such as "BFS_gDAC_1_stat.txt".
2. Compress the directory into a zip archive file named cse520-assgn03-LastName_FirstInitial.zip. Note that any object code or temporary files should not be included in the submission. Submit the zip archive to the course Canvas by the due date and time.
3. There will be 20 points penalty per day if the submission is late. Note that submissions are time stamped by Canvas. If you have multiple submissions, only the newest one will be graded. If needed, you can send an email to the instructor and TA to drop a submission.
4. The assignment must be done individually. No collaboration is allowed, except the open discussion in the forum on Canvas.
5. ASU Academic Integrity Policy (http://provost.asu.edu/academicintegrity), and FSE Honor Code (http://engineering.asu.edu/integrity) are strictly enforced and followed.

Important Notes:

- **Using fake data to draw table or graph and make comparisons will be treated as unethical practice and a violation of academic integrity will be reported.**
- **Any modifications to your submission or replacement of files will be handled as a new submission and may be subject to late submission penalty.**
- **Your submission should not contain any files of source code. All changes must be included in a patch file.**