18-640: Foundations of Computer Architecture

Project 1: Branch Prediction

Due: 11:59 pm EDT September 24, 2014

1. **Part 0 - Setup**
   Copy `/afs/ece.cmu.edu/class/ece640/project/project1/lab1.patch` in your gem5 folder and run,

   ```
   patch -p1 < lab1.patch
   ```

   Please make sure you see a list of file changes in `stdout`. This patch applies minor changes to the original release to make it easier for you to get started with Lab1.

2. **Part 1 – Analysis of Tournament and Bi-mode Predictor: (15 Points)**
   Read in detail about the Alpha 21264 Tournament Branch Predictor and Bi-mode Predictor.

   Alpha 21264 Tournament Predictor:
   Kessler, Richard E. "The alpha 21264 microprocessor." *Micro, IEEE* 19.2 (1999): 24-36.
   Bi-mode Predictor:
   Lee, Chih-Chieh, I-CK Chen, and Trevor N. Mudge. "The bi-mode branch predictor." *Microarchitecture, 1997. Proceedings., Thirtieth Annual IEEE/ACM International Symposium on*. IEEE, 1997.

   Go through gem5 source code for predictors at `src/cpu/pred/*`. Please understand the various functionalities implemented and relate it with the theory you have read.

   Your task in this part is to modify various parameters related to the two branch predictors and analyze the reason for the impact due to the changes.

   Remember, we'll be using the simulator in syscall emulation (SE) mode for this project.
   Example command that you will be implementing:

   ```
   build/X86/gem5.opt configs/example/se.py -c binary [-o options] --cpu-type=detailed --caches --pred-type=tournament --local-pred-size=4096 --global-pred-size=8192 --choice-pred-size=8192
   ```

   Explanation:
   - `build/X86/gem5.opt` - gem5 binary
   - `configs/example/se.py` - simulation script
   - `-c binary` - benchmark binary to run in SE mode
   - `-o options` - cmd line options for the binary
   - `--cpu-type=detailed` - specify cpu type (In Order, Out of Order etc.)
   - `--caches` - caches enable

We added following options to make it easier for you to change predictor type and their corresponding sizes at the command line, eliminating the need to recompile gem5 binary.

- `--pred-type=tournament` – branch predictor to be implemented
- `--local-pred-size=` local predictor size
- `--global-pred-size=` global predictor size
- `--choice-pred-size=` choice predictor size

Please see Appendix at the end of the handout for examples.

`BranchPredictor.py` describes the configuration parameters pertinent to branch prediction. You can configure type and size for each predictor here. Note that all the parameters are not used in each predictor.

After simulation, the stats will be generated in `m5out/` folder.

`.config` file is the configuration of the simulation performed

`.stat` file is the statistics generated at the end of simulation. You will need to extract info from this to analyze predictors.

Understand the `m5out/*.stat` file generated; following portion of the file is specifically generated for branch predictors:

```
system.cpu.branchPred.lookups                18211747
system.cpu.branchPred.condPredicted          17199049
system.cpu.branchPred.condIncorrect           6188168
system.cpu.branchPred.BTBLookups             10359986
system.cpu.branchPred.BTBHits                 8157880
system.cpu.branchPred.BTBCorrect                    0
system.cpu.branchPred.BTBHitPct             78.744122
system.cpu.branchPred.usedRAS                  174617
```

Calculating prediction rate:

```
100 - [(sys-
tem.cpu.branchPred.condIncorrect/system.cpu.branchPred.condPredicted)*100]
```

IPC can be found by looking for `system.cpu.ipc`

Your Task: Analyze predictors and report branch prediction rate and IPC for the following configurations only for *jpeg-encode* benchmark:
(Take number of counter bits as 2 for all data structures)

Tournament Predictor:

	Config – 1	Config – 2	Config – 3	Config – 4
Local Predictor Size	2048	4096	4096	4096
Global Predictor Size	8192	4096	8192	8192
Choice Predictor Size	8192	8192	4096	8192

Bi-mode Predictor:

	Config – 1	Config – 2	Config – 3	Config – 4
Global Predictor Size	2048	4096	8192	8192
Choice Predictor Size	4096	8192	4096	8192

Note: You are encouraged to try out different configurations (apart from the ones mentioned) and other benchmarks to do better analysis.

3. Part 2 – Implement gshare and YAGS Predictor (25+25 Points):

In this part you will be implementing the gshare branch predictor and YAGS branch predictor. You should refer to the following papers:

Gshare:

Scott McFarling, "Combining Branch Predictors" - Technical Note, *http://www.hpl.hp.com/techreports/Compaq-DEC/WRL-TN-36.pdf*

YAGS Predictor:

Eden, Avinoam N., and Trevor Mudge. "The YAGS branch prediction scheme."*Proceedings of the 31st annual ACM/IEEE international symposium on Microarchitecture*. IEEE Computer Society Press, 1998.

We have added dummy files such as {`gshare.cc, gshare.hh`} and {`yags.cc, yags.hh`} for you. You may take help from the existing predictors implemented in the distribution.

Your Task: Similar to the previous part, analyze using *jpeg-encode* and report results for the following configurations:

Gshare Predictor:

	Config – 1	Config – 2	Config – 3	Config – 4
Local Predictor Size	2048	4096	8192	16384

YAGS Predictor:

	Config – 1	Config – 2	Config – 3	Config – 4
Global Predictor Size	2048	4096	8192	8192
Choice Predictor Size	4096	8192	4096	8192

Note: Direct mapped cache for YAGS predictor would suffice for this part of the project.

4. **Part 3 – Challenge (15 Points):**
   Improve your predictor (gshare and YAGS) keeping the configuration:

   Gshare Predictor:

	Config
Local Predictor Size	4096

   YAGS Predictor:

	Config
Global Predictor Size	4096
Choice Predictor Size	8192

   You will be graded based on your *Branch Misprediction* (average of gshare and YAGS) relative to the rest of the class. It will be tested on a subset of MiBench benchmarks.
   Specifically, If you are in the best quartile, you will receive the full 15 of 15 points. If you are in the second best quartile, you will receive 10 of 15 points. If you are in the third best quartile, you will receive 5 of 15 points. If you are in the lowest quartile, you will receive 0 of 15 points.

   An exception is that you will receive the score of a better quartile if your branch misprediction rate is within 0.2% of the highest misprediction rate in that quartile.

   Note:
   You are allowed to deviate from the baseline design subject to the following restrictions.

   - If its gshare, it should have a global history register and xor-ing operation, or if its YAGS, it should have taken-not taken cache, choice PHT, global history register and no other data structure.
   - If you employ associativity, it must be 8-way or less.
   - Extravagant logic functions (such as dividing/multiplying by non-constants or non-powers of 2) are not allowed.

   If you have any doubt, feel free to ask us!

5. **Report (15 Points):**
   The report should present your branch predictor design and document the results of all branch predictors you analysed. Pay special attention to highlight the changes you tried and how effective they were. Finally, the report must address the following questions.

   1. Which predictor you analysed was the best? And Why?
   2. What is aliasing? Describe cases where aliasing is desirable and undesirable.
   3. Does increasing the number of bits in a saturation counter always improve performance? Give an example to support your position.
   4. Is the use of *BPHistory* structure in the predictor implementation required? State reasons.

   Feedback (5 points): Please provide us with a short feedback for this lab.

   Note: Please use pdf or plain text file format only.

6. **Hand-in Details:**
   Part 3 is the optimized version of you what you implemented in Part 2, so you just need to hand-in the optimized version.
   You will need to hand-in
   `src/cpu/pred/gshare.*`
   `src/cpu/pred/yags.*` and
   `report.pdf` or `report.txt` to
   `/afs/ece/class/ece640/submission/<andrew_id>/mygroup/project1/`

7. **Appendix:**

   Sample list of commands you may try running:
   (Append size parameters to commands appropriately)

```
#consumer/jpeg-encode
build/X86/gem5.opt --stats-file=jpeg-encode.stat --dump-config=jpeg-encode.config con-
figs/example/se.py -c mibench/consumer/jpeg/jpeg-6a/cjpeg -o "-dct int -progressive -opt -
outfile mibench/consumer/jpeg/output_small_encode.jpeg
mibench/consumer/jpeg/input_small.ppm" --cpu-type=detailed --caches

#consumer/jpeg-decode
build/X86/gem5.opt --stats-file=jpeg_decode.stat --dump-config=jpeg_decode.config con-
figs/example/se.py -c mibench/consumer/jpeg/jpeg-6a/djpeg -o "-dct int -ppm -outfile
mibench/consumer/jpeg/output_small_decode.ppm mibench/consumer/jpeg/input_small.jpg" --
cpu-type=detailed --caches

#automotive/qsort
build/X86/gem5.opt --stats-file=qsort.stat --dump-config=qsort.config con-
figs/example/se.py -c mibench/automotive/qsort/qsort_small -o
"mibench/automotive/qsort/input_small.dat" --
output=mibench/automotive/qsort/output_small.txt --cpu-type=detailed --caches

#network/dijkstra
build/X86/gem5.opt --stats-file=dijkstra.stat --dump-config=dijkstra.config con-
figs/example/se.py -c mibench/network/dijkstra/dijkstra_small -o
"mibench/network/dijkstra/input.dat" --output=mibench/network/dijkstra/output_small.dat --
cpu-type=detailed --caches

#office/stringsearch
build/X86/gem5.opt --stats-file=strsearch.stat --dump-config=strsearch.config con-
figs/example/se.py -c mibench/office/stringsearch/search_small --
output=mibench/office/stringsearch/output_small.txt --cpu-type=detailed --caches

#telecomm/FFT
build/X86/gem5.opt --stats-file=fft.stat --dump-config=fft.config configs/example/se.py -c
mibench/telecomm/FFT/fft -o "4 4096" --output=mibench/telecomm/FFT/output_small.txt --cpu-
type=detailed --caches

#telecomm/FFT-inv
build/X86/gem5.opt --stats-file=fft_inv.stat --dump-config=fft_inv.config con-
figs/example/se.py -c mibench/telecomm/FFT/fft -o "4 8192 -i" --
output=mibench/telecomm/FFT/output_small.inv.txt --cpu-type=detailed --caches
```