High Performance Computer Architecture (CS60003)

Assignment 1

Indian Institute of Technology Kharagpur

In this assignment, you are required to use the `gem5` simulator to study and analyze the effects of varying Branch Predictors on a given processor model using a benchmark program. *A tutorial session has already been provided on how to install and setup gem5.* The detailed slides are available on Moodle. For more information on gem5, please visit `http://learning.gem5.org/book/`

The aim of this assignment is to make you familiar with the effects of Branch Predictors on the performance of a system. There are a number of Branch Predictors available in gem5, which can be found in "`$gem5/src/cpu/pred`". The assignment is composed of four different modules.

1. **Adding Branch Prediction Support to TimingSimple CPU**: gem5 uses a number of CPU models, like `AtomicSimple, TimingSimple, InOrder,` etc. The `TimingSimpleCPU` is the version of SimpleCPU that uses timing memory accesses. By default, `TimingSimpleCPU` does not have branch predictor support. You need to add the support of the three branch predictors, such as, `2bit_local, Bi_mode and Tournament` Predictors (separately) to `TimingSimpleCPU`. Run a HelloWorld program, as shown in the tutorial (in class). This should generate 'configs.ini' file in "`m5out`" folder in the `$gem5` directory. If your run was successful, you should be able to check the BranchPredictor being used in the config.ini file. A snapshot of a sample 'config.ini' file is shown below.

```
gem5-stable/m5out$ vi config.ini

86 [system.cpu.branchPred]
87 type=BiModeBP
88 BTBEntries=4096
89 BTBTagSize=16
90 RASSize=16
91 choiceCtrBits=2
92 choicePredictorSize=8192
93 eventq_index=0
94 globalCtrBits=2
95 globalPredictorSize=8192
96 instShiftAmt=2
97 numThreads=1
```

   Deliverable: Three 'configs.ini' files. One for each branch predictor.

2. **Adding extra Resulting Parameters in the stats file**: For every successful run, a 'stats.txt' file is generated in `m5out` folder which shows different statistics like `final_ticks, cache_hits, cache_misses`, etc. In this part, you are required to add two more parameters to the 'stats.txt' file. Note that the stats file is automatically generated and the event names and its corresponding values are generated by different source files in gem5. Run a HelloWorld program, similar to the previous part, with any of the three above-mentioned BranchPredictor support. The two parameters that you need to add are -

   (a) **BTBMissPct = (1 - (BTBHits/BTBLookups))** $\times 100$
   where: `BTBHits -> total number of BTB Hits`
   `BTBLookups -> total number of BTB references`

   (b) **BranchMispredPercent = (numBranchMispred ÷ numBranches)** $\times 100$
   where: `numBranchMispred -> total number of mispredicted Branches`
   `numBranches -> total number of branches fetched`

   Below is a snapshot of a sample 'stats.txt' file with the modification:

```
./stats.txt:system.cpu.branchPred.lookups              1317       # Number of BP lookups
./stats.txt:system.cpu.branchPred.condPredicted        1317       # Number of conditional
branches predicted
./stats.txt:system.cpu.branchPred.condIncorrect         621       # Number of conditional
branches incorrect
./stats.txt:system.cpu.branchPred.BTBLookups            203       # Number of BTB lookups
./stats.txt:system.cpu.branchPred.BTBHits               202       # Number of BTB hits
./stats.txt:system.cpu.branchPred.BTBCorrect              0       # Number of correct BTB
predictions (this stat may not work properly.
./stats.txt:system.cpu.branchPred.BTBHitPct        99.507389       # BTB Hit Percentage
./stats.txt:system.cpu.branchPred.BTBMissPct        0.492611       # BTB Miss Percentage
./stats.txt:system.cpu.branchPred.usedRAS                 8       # Number of times the
RAS was used to get a target.
./stats.txt:system.cpu.branchPred.RASInCorrect            0       # Number of incorrect
RAS predictions.
./stats.txt:system.cpu.Branches                        1317       # Number of branches
fetched
./stats.txt:system.cpu.predictedBranches                210       # Number of branches
predicted as taken
./stats.txt:system.cpu.BranchMispred                    621       # Number of branch
mispredictions
./stats.txt:system.cpu.BranchMispredPercent        47.152620       # Percent of Branch
Mispredict
```

This will require you to modify in the following source files -
`$gem5/src/cpu/simple/exec_context.hh`
`$gem5/src/cpu/pred/bpred_unit.hh`
`$gem5/src/cpu/pred/bpred_unit.cc`
`$gem5/src/cpu/simple/base.cc`

Note that everytime you modify something in the source files, you have to build gem5 (using `scons build/X86/gem5.opt -j5`).

Deliverable: The above four files and 'stats.txt' from `md5out` folder

3. **Implement a Gshare Predictor**: In this part, you are required to implement Gshare branch predictor. 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. The branch predictor in gem5 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**. To implement a new branch predictor, you only have to fill up these 5 function with your own design. 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.
   - **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 following table contains the parameters you need to set for the three branch predictors. Run using the **se.py** script provided in gem5. You only enable the following flags and leave all others as the default `--cpu-type=DerivO3CPU --caches --l2cache --bpred= TournamentBP(), LocalBP()` and `GshareBP()`.

2-bit local	Tournament	Gshare
16384 entries * 2 bits	Local History table - 1024 * 11 bits Local predictor size - 2048 * 2 bits One 12-bit Global history register Global predictor size - 4096 * 2 bits Choice predictor size - 4096 * 2 bits	16384 entries * 2 bit one 16-bit Global history register

Deliverables: Submit only the modified files.

4. **Running the benchmarks for comparison**: In this part, you are required to compare between the performance of three branch predictors - `2bit_local, Tournament, Gshare` (that you just implemented) using a benchmark program. The benchmark program named `blocked_matmul.c` has been provided in Moodle. Run the program using **se.py** script with the following cache parameters: `--l1d_size=128kB --l1i_size=128kB --l2_size=1MB --l1d_assoc=2 --l1i_assoc=2 --l2_assoc=4 --cacheline_size=64`.

   Deliverables: A report (in a pdf format) about the observed effects of the Branch Predictors with respect to the benchmark program.

   N.B. All the submissions will be screened through Moss (`https://theory.stanford.edu/~aiken/moss/`). If considerable similarities are found between multiple submissions, they will be heavily penalised.