Project #2

(Report due is 11:59:59 PM, 10/13/2022)

Project description:

In this project, we will explore the design of branch predictor in compute system. Specifically, we will introduce how to add branch predictor module and change its attributes in configuration files. We also provide a tutorial to add new branch predictor strategy in Gem5 simulator. You will evaluate how different types of branch predictors and their configurations can impact the system performance. In deliverable, you are required to find out the best branch predictor configurations to maximize the performance of different workloads. Whenever you face the trouble or problem on this project, please refer to Gem5 document here or contact TAs in WeChat. All the following information have been verified on Ubuntu 20.04.1 with Linux Kernel 5.15.0.

Submission:

- Please submit (in a report form, both PDF and Word are acceptable) the deliverables for each part in order and clearly defined. Please give a brief description of the results in your report and DO NOT SUBMIT ANY SCRIPTS.
- Please send your report as an attachment at ca2022fall@163.com. The titles of your email AND attachment should both be Student ID_Name_Proj2 (e.g., 123456789_Nahida_Proj2).
- DO NOT PLAGIARIZE. After some projects, we will select 10 students randomly and ask them to run their codes in person and answer our questions related to their codes.

Late policy:

- You will be given 3 slip days (shared by all projects), which can be used to extend project deadlines, e.g., 1 project extended by 3 days or 2 projects each extended by 1 days.
- Projects are due at 23:59:59, no exceptions; 20% off per day late, 1 second late = 1 hour late = 1 day late.

Part 1: Add branch predictor in the configuration script

In this section, we will extend the configuration script two-level.py in project #1 by adding different branch predictor objects. Let's start with create a new folder proj1 and copy configuration files new_cache.py and two-level.py from proj1 to proj2.

```
sudo apt-get update; sudo apt-get upgrade

mkdir configs/proj2

cp configs/proj1/new_cache.py configs/proj2/.

cp configs/proj1/two-level.py configs/proj2/.

vi configs/proj2/two-level.py
```

Then, we should change the default CPU type to "O3CPU". This is because SimpleTimingCPU cannot reflect the performance impact of branch predictor as it assumes each instruction takes single CPU clock cycle. This can be done by inserting the command line as follows:

```
if options.o3:
    root.system.cpu = DerivO3CPU()
elif options.inorder:
    root.system.cpu = MinorCPU()
else:
    root.system.cpu = TimingSimpleCPU()
    root.system.cpu = DerivO3CPU()
```

Gem5 provides different types of branch predictors, in this project, we focus on three of them: local branch predictor, tournament branch predictor, and BiMode branch predictor. Any of these three branch predictor classes can be used to instantiate CPU's branch predictor object. In addition, you are able to configure each branch predictor with different configuration parameters. To avoid the overhead of modifying configuration script, it's recommended to add command line parameters for branch predictor. The following commands enable all the command line parameters that will be used in the evaluation:

```
parser.add_option('--local', action="store_true")
parser.add_option("--tournament", action="store_true")
parser.add_option('--bimode', action="store_true")
parser.add_option('--btbentry', type="int", default=4096)
parser.add_option('--ras', type="int", default=16)
parser.add_option('--localsize', type="int", default=2048)
parser.add_option('--localhissize', type="int", default=2048)
parser.add_option('--globalsize', type="int", default=8192)
parser.add_option('--choicesize', type="int", default=8192)

########################## add branch predictor ############################
if options.local:
    root.system.cpu.branchPred = LocalBP()
elif options.tournament:
    root.system.cpu.branchPred = TournamentBP()
elif options.bimode:
    root.system.cpu.branchPred = BiModeBP()
```

```
else:
    root.system.cpu.branchPred = TournamentBP()


root.system.cpu.branchPred.BTBEntries = options.btbentry
root.system.cpu.branchPred.RASSize = options.ras


if options.local:
    root.system.cpu.branchPred.localPredictorSize = options.localsize
elif options.tournament:
    root.system.cpu.branchPred.localPredictorSize = options.localsize
    root.system.cpu.branchPred.localHistoryTableSize = options.localhissize
    root.system.cpu.branchPred.globalPredictorSize = options.globalsize
    root.system.cpu.branchPred.choicePredictorSize = options.choicesize
elif options.bimode:
    root.system.cpu.branchPred.globalPredictorSize = options.globalsize
    root.system.cpu.branchPred.choicePredictorSize = options.choicesize
############################ add branch predictor ############################
```

"--local", "--tournament", "--bimode" enable local branch predictor, tournament branch predictor, and BiMode branch predictor in your simulation, respectively. While each branch predictor provides multiple parameters for users, we enumerate 6 parameters which have the most impact on the performance. All the branch predictors leverage the same infrastructures such as branch target buffer (BTB) and return address stack (RAS). The BTB size and RAS size can be changed via "--btbentry" and "--ras". You also can configure local predictor size ("--localsize"), local history table size ("--localhissize"), global predictor size ("--globalsize"), and choice predictor size ("--choicesize").

Part 1 deliverables:

1. Modify the branch predictor types as local branch predictor, tournament branch predictor, and BiMode branch predictor. You need to compare the performance of these branch predictors in the 2MM and BFS workloads we provide in project #1 package and write your own analysis with bar graphs in your reports. When you compare performance, you can use IPC (instruction per cycle) or branch predictor hit rates in the stats.txt file. Please configure the 6 parameters ("--btbentry", "--ras", "--localsize", "--localhissize", "--globalsize", "--choicesize") with default values.
2. Evaluate different BTB sizes ("--btbentry") on each branch predictor and 2MM and BFS workloads. You need to compare the performance and analyze the simulation results with bar graphs in your reports.

3. Evaluate different RAS sizes ("--ras") on each branch predictor and 2MM and BFS workloads. You need to compare the performance and analyze the simulation results with bar graphs in your reports.
4. Evaluate different local predictor sizes ("--localsize") on only local branch predictor and BFS workload. You need to compare the performance and analyze the simulation results with bar graphs in your reports.
5. Evaluate different global predictor sizes ("--globalsize") on only BiMode branch predictor and BFS workload. You need to compare the performance and analyze the simulation results with bar graphs in your reports.
6. Evaluate different local predictor sizes ("--localsize"), local history table sizes ("--localhissize"), and global predictor size ("--globalsize") on only tournament branch predictor and BFS workload. You need to compare the performance and analyze the simulation results with bar graphs in your reports.
7. Please search for information and describe the designs of tournament branch predictor and BiMode branch predictor, then analyze the pros and cons of them.

For simplicity, when you evaluate the impact of one parameter on the performance (e.g., localsize), you can set other parameters to default (e.g., localhissize and globalsize). Moreover, there is no need to set step size too small. Evaluating every parameter with 3~5 values is enough.

Appendix A

```
process.cmd = ['test_bench/2MM/2mm_base']

process.cmd = ['test_bench/BFS/bfs','-f','test_bench/BFS/USA-road-d.NY.gr']

process.cmd = ['test_bench/bzip2/bzip2_base.amd64-m64-gcc42-nn','test_bench/bzip2/input.source','280']

process.cmd = ['test_bench/mcf/mcf_base.amd64-m64-gcc42-nn','test_bench/mcf/inp.in']
```