CS425: Computer Systems Architecture

Programming Assignment 2
Assignment Date: 23/12/2021
Due Date: 14/01/2022 23:59

Instructions: Send your deliverable zip files via e-mail to HY425 course e-mail (hy425@csd.uoc.gr). Set the e-mail subject: HY425 – Programming Assignment 2

The Limits of ILP

The purpose of this assignment is to introduce you in architectural exploration using simulation and help you familiarize with some of the details of Out-of-Order CPU by exploring some of its design points and potential performance improvements. You have to run simulations with different parameters (the actual list of parameters can be found below) and see the impact of the alternative design choices in terms of Instructions Per Cycle (IPC). Note at this point, that you are free to refer to other metrics, as well.

For your simulations you will use the gem5 simulator (www.gem5.org/). Although the assignment does not require prior experience with gem5, it is recommended that you take some time to follow the gem5 tutorial:

www.gem5.org/documentation/learning_gem5/introduction/

Simulator

In this assignment you will compile gem5 to target on the x86 ISA. The build requirements can be found here: www.gem5.org/documentation/learning_gem5/part1/building/
You are able to work on this assignment solely on your own environment.
You can get a snapshot of the simulator by cloning this repository:

https://github.com/sotot0/cs425_PA2.git

The repository contains:
i. the gem5 directory,
ii. and the benchmark directory

After you have installed the indicated dependencies, clone the repository and compile the simulator by typing:

```
python3 `which scons` build/X86/gem5.opt -j9
```

inside the gem5 directory. The compilation step consumes time so complete it as soon as possible!

To run a batch of five (5) benchmarks with the simulator you have to use the following command inside the benchmark directory:

```
./runall.sh [List of Arguments]
```

This command will run all the benchmarks while simulating a setup with an Out-Of-Order CPU (O3CPU), a good performing branch predictor (TournamentBP) and reasonable settings for caches and other components. You are encouraged to examine thoroughly the runall.sh / README files in order to figure out the different arguments that you will be allowed to change. The arguments that are related to the branch predictor (TournamentBP) are not allowed to change. Also, arguments that are related to the cache and CPU domain are not allowed to change during the assignment. Note that the -I *[Instruction Count]* argument means that for each benchmark, gem5 simulates *[Instruction Count]* instructions. You can change this parameter during early testing but for generating final numbers you should have at-least 50 million instructions. If you set a very large number of instructions the simulation time will increase dramatically. (Even with the parameter set to 50 million instructions you may experience long simulation times especially when running benchmark1/benchmark4). Ensure that your comparisons are based on the same instruction count between different benchmarks and/or settings to extract uniform and fair results. Also, ensure that this parameter is large enough to extract meaningful results. An estimated number would be between 50-80 million instructions. In any case, this is another reason to begin the assignment as soon as possible!

Explore OoO CPU Design Points

Your main task is to explore the given design space of an OoO CPU by simulating benchmarks and observe mainly the corresponding IPC values. Study the slides presented in the class and the paper from David W. Wall: "Limits of Instruction-Level Parallelism" - you can find it in the website of the course.

The list of the specified design parameters is the following:

- LSQ_ENTRIES: Number of LoadQueue/StoreQueue entries (same value for both)
  - Range: 8 – 512
  - Default: 8
  - Today: up-to 128
- ROB_ENTRIES: Number of Reorder Buffer entries
  - Range: 16 – 2048
  - Default: 16
  - Today: up-to 512
- FUINT_NUM: Number of Simple Int Functional Units (ADD etc.)
  - Range: 1 – 16
  - Default: 2
  - Today: up-to 5
- FUCOM_NUM: Number of Complex Int Functional Units (DIV, MULT etc.)
  - Range: 1 – 16
  - Default: 2
  - Today: up-to 2
- FUFP_NUM: Number of Floating Point Function Units (FADD etc.)
  - Range: 1 – 16
  - Default: 2
  - Today: up-to 3
- FULS_NUM: Number of Load/Store Functional Units (same value for both)
  - Range: 1 – 16
  - Default: 1
  - Today: up-to 2

1. Explore the IPC improvements using the ranges of these parameters compared to the default values (select your sweep steps carefully!) and find the combination of parameter values with the minimum product MiP (i.e. MiP = LSQ_ENTRIES ✕ ROB_ENTRIES ✕ FUINT_NUM ✕ FUCOM_NUM ✕ FUFP_NUM ✕ FULS_NUM) that maximizes IPC using both realistic and non-realistic ranges. Explain your methodology.
2. Is there a saturation point of each benchmark's IPC? Explain your answers.
3. Provide your answers in a report in PDF format and use graphs and/or tables.

It is important to get familiar with the generated m5out directories under /benchmarkN directories after each run. These directories include files showing the simulated configuration and their statistical information.

Bonus Question

Use the generated statistics (e.g. instruction mix) to infer information and/or other characteristics about the behavior of each benchmark. Explain your approach and observations.

Deliverables

You have to deliver one **zip/tar.gz** with the following files:

- Your report in PDF format
- Any custom scripts that you used
- Any other files that you find useful
- m5out directories that contain useful statistics along with the corresponding configurations