ECE 462/562
Homework 2

Due: Friday, March 20, 11.59pm.

- *Work in your project groups. Assign benchmarks to different group members. Each group member must run simulations for at least one of the benchmarks. Only one submission per group is required.*
- *This homework assumes some basic knowledge of Linux commands (e.g., cd, tar, etc.) If you don't know how to do something, Google it, or interact with your colleagues.*

1. In-order vs. Out-of-Order Execution

In this part of the assignment, you will explore the performance benefits of in-order vs out-of-order (OoO) execution using the ARM big.LITTLE models implemented in GEM5. The big core is an OoO processor that incorporates dynamic scheduling as discussed in class, while the LITTLE core is in-order.

Using the six provided benchmarks, run simulations as follows:

```
build/ARM/gem5.fast    --stats-file=<$statsFileName>    --dump-config=<$location/config.ini>
configs/example/se.py            --caches            --l1i_size=32kB            --l1i_assoc=4
--l1d_size=32kB --l1d_assoc=4 --cacheline_size=64 --l2cache --l2_size=1MB --l2_assoc=8 --cpu-
clock=1.6GHz --cpu-type=<$cpuType> -n 1 --maxinsts=100000000 --bench <$benchmark>
```

Where <cpuType> = ex5_LITTLE and $cpuType = ex5_big for the LITTLE and big cores, respectively. (Note that these options are case-sensitive).

For each benchmark, fill in part 1 of homework2_deliverables.docx with the required information comparing in-order and out-of-order cores.

2. Reorder Buffer (ROB) Impact

While OoO may improve performance for several applications, it also introduces hardware overhead. For instance, the reorder buffer (ROB), which is used to ensure in-order completion, can introduce area and energy overhead.

In the OoO system, we would like to explore whether or not we can use a smaller ROB without substantially degrading the performance gains from OoO execution. The default ROB has 60 entries.

For the provided benchmarks, using the default (60-entry ROB) and cpu-type=ex5_big as the base configuration, compare the execution time changes of using 20-, 40-, and 80-entry ROB, while keeping all other configurations constant.

The number of ROB entries can be found in *gem5/configs/common/cores/arm/ex5_big.py* within *class ex5_big(DeriveO3CPU)* (around line 147). In general, there is no need to rebuild GEM5 after changing Python configurations.

Represent your performance comparison using a bar chart. Plot the execution times of the 20-, 40-, and 80-entry ROBs normalized to the 60-entry ROB. You may use MS Excel or any tool you're familiar with to create the graph. Copy and paste your graph into part 2 of *homework2_deliverables.docx*.

3. Load/Store Queue (L/SQ) Impact

Now, perform the same experiments for the load and store queues and a static ROB of 60 entries. The default in GEM5 is 16 entries for both the load and store queues. Explore the impact of the different LSQs using the 16-entry LQ and SQ as the base, and varying the sizes to 4-entry, 8-entry, and 32-entry.

The number of LQ/SQ entries can be found in *gem5/configs/common/cores/arm/ex5_big.py* within *class ex5_big(DeriveO3CPU)* (around lines 110 and 111). Keep both LQ and SQ entries equal for all the simulations.

Represent your performance comparison using another bar chart. Plot the execution times of the 4-, 8-, and 32-entry LSQs normalized to the 16-entry LSQ. You may use MS Excel or any tool you're familiar with to create the graph. Copy and paste your graph into part 3 of *homework2_deliverables.docx*.

Remember to convert the deliverables file to pdf before submitting. Only pdf files will be graded!