Project2: GPGPU-SIM

1 Objectives

The objective is to get you started with gpgpu-sim simulator. This assignment is meant to familiarize you with the simulator for PA3 and final project.

2 Ground Rules

(1) All students must work alone. Sharing code between students will be considered as cheating and receive appropriate action based on the University Policy. (2) A Q&A forum has been created on Moodle for posting questions, discussing and debating issues, but please do not share your code directly on Moodle.

(3) This assignment has different parts for you to learn the functional simulator and performance simulator about GPGPU-Sim, **and each part are independent.**

(4) **Every student should follow the submission requirement exactly, otherwise you will receive points deduction.**

3 Description

Each task are independent, when you finish one task, please comment the code you have modified firstly before you move to the next.

3.1 Functional Simulator

3.1.1 Task1: Instruction semantic

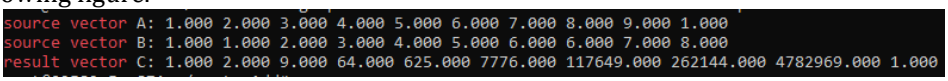
This task requires you to modify the functional simulator. You need to change the semantics of float(32-bits) point **add** instruction to **power** operation. Suppose you have a float point add instruction fd = fa + fb. The register fa and fb store 32-bits float point value. fa =2.0, fb=3.0, the original add instruction will generate the new value 5.0 and store it to fd. Now the new semantic should generate the value 8.0 = $2.0^{3.0}$ and store it to fd. For this task, the test case is vectorAdd, we have uploaded one vector add CUDA program on moodle.

Please use this one to test your code. The original instruction semantic should generate the results as follows.

```
source vector A: 1.000 2.000 3.000 4.000 5.000 6.000 7.000 8.000 9.000 1.000
source vector B: 1.000 1.000 2.000 3.000 4.000 5.000 6.000 6.000 7.000 8.000
result vector C: 2.000 3.000 5.000 7.000 9.000 11.000 13.000 14.000 16.000 9.000
```

You can verify this result(original semantic) by run the vector add code on Hydra.

The new instruction semantic should generate the results shown in the following figure.

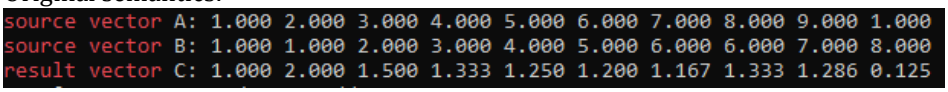


You can verify the new semantic by running your code on GPGPU-SIM.

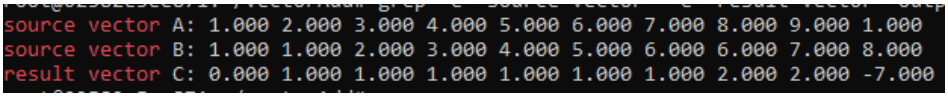
We will use two test case, one is vectorAdd provided on Moodle, another one would be similar to vectorAdd but used as secrete test case.

Also change the semantics of floating point(32 bits) **divide** to **subtract** operation. Suppose you have a float point divide instruction fd = fa / fb. The register fa and fb store 32-bits float point value. fa =6.0, fb=2.0, the original divide instruction will generate the new value 3.0 and store it to fd. Now the new semantic should generate the value 4.0 =6.0- 2.0 and store it to fd.

Original semantics:



New Semantics:



Hints, for this task, you need to read the code in **instructions.cc** under the cuda-sim directory carefully and understand how it works. In your report, have a short explanation about your understanding and where you have modified. **Note:** Follow the same steps you used to pull the vectorAdd github file and just replace the cuda file with the cuda file given on Moodle.

3.2 Performance/Timing simulator

This task you need to work on performance/timing simulator. The **maximum instructions** you should simulate is **100000000** for the following tasks. For those benchmarks have less than 100000000 instructions, just let it run until it finished.

3.2.1 Task2: Benchmark performance study

In this task, you need to chosse 6 benchmarks from ispass-2009 and run them In this task, you need to use the 5 benchmarks from ISPASS.zip and run them with SM6_TITANX and SM7 TITANV configurations and plot the IPCs of these benchmarks. This study is helping you get familier with benchmarks cause you will use these benchmarks in your final project.

Please download the compiled benchmarks on moodle. To run the benchmarks, you can follow the instruction listed here.

a. create ”test” directory under gpgpusim (mkdir test).
b. copy the SM6 _TITANX and SM7 TITANV to ”test” directory.
c. copy the benchmark to ”test” directory.
d. under each benchmark, create two soft links that point to gpgpusim.configand config XX islip.icnt under SM6 TITANX and SM7 TITANV(using ln command), or just copy these two files to each benchmarks directory. e. run the benchmark, please pay attention to the benchmark input, the input data has been provided with the benchmarks. Do not delete these files.

Graph: Plot IPCs for each benchmarks, X-axis: benchmarks, Y-axis: IPCs. You should have two lines for SM6 TITANX and SM7 TITANV respectively. Have a short discussion about your plots.

3.2.2 Task 3: Branch instructions and divergence

For this task, you need to modify the performance/timing simulator code by adding counters to report the following statistics.
(1)total number of warps that executed conditional branch instructions and have divergence.
(2)total number of warps that executed conditional branches, no matter they have divergence or not.
Hints: For this task, please read the scheduler unit in shader.cc and the simt stack in abstract hardware model carefully and understand how it works. The benchmark you should use for this task is LPS. You need to dump out the counters and report the counters in your report. The format you dump out the counters is shown as follows

```
# of warps excuted conditional branch instrctions and have divergence: XXX

# of warps excuted conditional branch instrctions(no matter they have divergence or not): XXX
```

3.2.3 Task 4: Memory access space

For this task, you need to modify the performance/timing simulator code by adding a counters to report the following statistics.
(1)The total number of global memory access.
(2)The total number of local memory access.
Hints: For this task, you need to read the code about load-store unit in share.ccFor this task, you need to read the code about load-store unit in shader.cc carefully and understand how it works. You need to dump out the counters and report the counters in your report. The benchmark you should use for this task is LPS. The format you dump out the counters is shown in the following figure.

```
# of global memory access: XXX

# of local memory access: XXX
```

4 Submission

(1)Create a directory named ”cuda-sim” and copy the files you modified for task1 to this directory.
(2)Create a directory namde ”gpgpu-sim”, then create two directories named ”task3” and ”task4” and copy the files you modified for task3 and task4 respectively
(3)Write the report, for task1, task3 and task4, please have a short discussion about where you modified. For task2, including all the plots and your analysis in your report.
(4)zip ”cuda-sim”, ”gpgpu-sim” and your report as unity id.zip

5 Grading

(1)task1: 25 points, 10 points for each test case(totally 2 test case), 5 points for discussion in your report
(2)task2: 20 points, 15 points for the plots, 5 points for discussion
(3)task3: 30 points, 5 points for compiling your code successfully, 20 points for dumping out the counters correctly, 5 points for reporting the counters in your report.
(4)task4: 25 points, 5 points for compiling your code without any errors, 15 points for dumping out the counters correctly, 5 points for reporting the counters in your report.