Programming Assignment 3 Part B

1 Objective

Optimize the quantum circuit simulation process of applying six single-qubit gates to six different qubits in an n-qubit quantum circuit.

2 Ground Rules

1. All students must work alone. Sharing code between students will be considered 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. Every student should follow the submission requirement exactly, otherwise, you will receive a point deduction.

3 Description

In PA1, we have written the Cuda code for the quantum circuit simulation of applying a single-qubit quantum gate to one qubit in an n-qubit quantum circuit. In PA3, we want to write the Cuda code to do the quantum circuit simulation of applying six single-qubit quantum gates to six different qubits in an n-qubit quantum circuit. And we have provided the format of the input file and the expected output as shown in the next section.

Task1. Write the code to read in the input file and generate the expected output result of applying six single-qubit quantum gates to six different qubits in an n-qubit quantum circuit.

You can divide the simulation process of PA3 into six steps and apply one single-qubit gate at one step, and in this way, each step will be the same as the simulation process of PA1.

Task2. Use shared memory to optimize the above code you have written.

This optimization method is called ShareMem Method and comes from https://dl.acm.org/doi/pdf/10.1145/3447818.3460357. You can read section 3.1 from this paper to learn more about this ShareMem Method. Be careful that in this paper, the least significant bit is on the left most of the index. However, in PA1's simulation example, the least significant bit is on the rightmost of the index. And for the simulation process of PA3, we will keep the assumption that the least significant bit is on the rightmost of the index.

The basic idea of this ShareMem Method is can be divided into 3 steps as shown below:

- For an n-qubit quantum circuit that has an initial quantum state of 2^n values, split the 2^n values into several independent fragments of size 2^6, then map each fragment to one GPU thread block. For each thread block, load the 2^6 values of the fragment from global memory to shared memory. (The reason for using 2^6 as the fragment size is because there are six single-qubit gates applied to six different qubits.)
- Apply the six single-qubit gates on each of the fragments and get the results. Similar to applying 6 gates on a 6-qubit circuit, you can split this process into 6 steps: applying the first gate to qubit 0, applying the second gate to qubit 1, applying the third gate to qubit 2, applying the fourth gate to qubit 3, applying the fifth gate to qubit 4, and applying the sixth gate to qubit 5. And for each of the steps, the computation is the same as in PA1, and here since we have 2^6 values that need to be calculated in each thread block, there will be 2^5 threads in each thread block, one thread will do one matrix multiplication as shown in the figure below.
- After finishing the above computation and getting results, the 2^6 values of each fragment in each of the thread blocks are stored back in global memory.

$$\begin{pmatrix} a'_{b_{n-1},\ldots,b_{t+1},0,b_{t-1},\ldots,b_0} \\ a'_{b_{n-1},\ldots,b_{t+1},1,b_{t-1},\ldots,b_0} \end{pmatrix} = \begin{bmatrix} U_{0,0} & U_{0,1} \\ U_{1,0} & U_{1,1} \end{bmatrix} \begin{pmatrix} a_{b_{n-1},\ldots,b_{t+1},0,b_{t-1},\ldots,b_0} \\ a_{b_{n-1},\ldots,b_{t+1},1,b_{t-1},\ldots,b_0} \end{pmatrix}$$

Task3. (Optional, bonus points) Thread coarsening optimization

Use thread coarsening optimization to optimize the code in task2.

4 Input format and expected output

1. The submitted code will be tested using a randomly generated input file. The input format is very similar to the format of PA1. After you download the example of the input file from moodle, you can see that in the input file, there are three parts as shown below:

- There are a total of 6 matrices representing the 6 gates and they are separated by a blank line. The first 2x2 matrix represents the first single-qubit quantum gate, and second 2x2 matrix represents the second single-qubit quantum gate, the third 2x2 matrix represents the third single-qubit quantum gate, and so on. Each matrix row is in a separate line (i.e. ends with a linefeed "\n"), and matrix elements in the same row are separated by a single space.

- Below the 6 matrices, there is a 128x1 vector that represents a 7-qubit (from qubit 0 to qubit 6) quantum state. This vector size may change based on the circuit size, but you can assume that the circuit size will not be larger than 30 qubits, so this vector size will not be larger than 2^30.
- The last six numbers 0, 2, 3, 4, 5, 6 in the last six lines represent that the first gate is applied to qubit 0, the second gate is applied to qubit 2, the third gate is applied to qubit 3, the fourth gate is applied to qubit 4, the fifth gate is applied to qubit 5, and the sixth gate is applied to qubit 6. And these six numbers may change which means we are applying the six gates to different qubits of an n-qubit quantum circuit.

2. The expected output format is the same as PA1. You should output a vector of length N = 2^n, and n equals the size of the quantum circuit.

3. For grading, your program will be compiled and run with commands like "./quamsim ./inputfile_name.txt"

5 Submission Requirements

1. Submit the code for the task1 and the code for task2. And submit the code for task3 if you have done task3. Your code should work on both the hydra cluster and the GPGPU-SIM simulator.

2. In your report, for task1, briefly explain your CUDA kernel functions; for task2, explain how you optimize the code using the ShareMem method; and explain how you do the thread coarsening optimization for task3 if you have done the task3.

3. Use GPGPU-SIM to count the number of global memory accesses of task1 and task2, report the count number of task1 and task2 and analyze their differences in your report.

4. zip the code, Makefile, and the report as unity id.zip.

6 Grading

1. Task1: 35 points, 30 points for the code to work correctly, and 5 points for the discussion in your report.
2. Task2: 65 points, 50 points for the code to work correctly, and 15 points for the discussion in your report.
3. Task3: 15 points, 10 points for the code to work correctly, and 5 points for the discussion in your report.