Programming Assignment 1 - Quantum Gate Simulation

1. Task Description

We will simulate a "single-qubit quantum gate" applied to an N-bit quantum state. Despite the name of the assignment, you do not need to know anything about quantum gates or circuits. This document will explain how a quantum gate is simulated as a sequence of 2x2 matrix multiplications.

A single-qubit quantum gate operation can be simulated as a sequence of 2x2 matrix multiplications. Quantum gates are applied to qubits of quantum states, which can be any number of qubits long. An n-qubit quantum state is represented as a vector of length $N = 2^n$, i.e. $[a_0, a_1, a_2 .... a_{N-1}]$. The indices can also be represented with binary notation. For example, $a_9 = a_{0001001}$ in a 7-qubit system.

A single-qubit quantum gate can be modeled as a 2x2 matrix U. With a single-qubit quantum gate applied to qubit t, the output state $[a'_0, a'_1 ... a'_{N-1}]$, can be computed with $2n - 1$ matrix multiplication tasks. Each matrix multiply updates two positions in the output state, as seen below

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

In the equation above, $[b_{n-1} .... b_0]$ are the indices in the binary format. Applying a single-qubit gate over the t-th qubit b_t on the n-qubit system can be achieved by $2n - 1$ matrix multiplication tasks, where each task updates two positions with only the index t differing, and this t-th index is highlighted in red in the above image.

For example, consider applying a single-qubit gate on qubit 2 of a 7-qubit state (from qubit 0 to qubit 6). We can simulate this operation with 64 ($= 2^6$) matrix multiplications. Since qubit 2 is the third bit from right on this 7-qubit circuit, each matrix multiplication task will calculate two positions with only the third bit differing. Therefore, the first one calculates $a'_{0000000}$ and $a'_{0000100}$, the second one calculates $a'_{0000001}$ and $a'_{0000101}$, the third one calculates $a'_{0000010}$ and $a'_{0000110}$, the fourth one calculates $a'_{0000011}$ and $a'_{0000111}$, the fifth one calculates $a'_{0001000}$ and $a'_{0001100}$, the sixth one calculates $a'_{0001001}$ and $a'_{0001101}$, etc.

You need to develop the GPU kernel code as well as the host code to perform quantum gate simulation. Develop two versions of the host code, one uses `cudaMalloc` and `cudaMemcpy` to move data explicitly and the other uses `cudaMallocManaged` to leverage unified virtual memory to move the data. Finally, prepare a 1 page report briefly describing how your host code and the kernel work. Also report the timing results for both versions.

2. Constraints

Quantum state bits: **N, 4 <= N <= 16**
Input/output quantum state elements will be of **float** type.

3. Input format and expected output

The input to your program will be a single .txt file. In the .txt file, the 2x2 matrix at the beginning represents the single-qubit quantum gate, followed by 2^N float values which represent the input quantum state vector. The number in the last line of the input file represents which qubit the single-qubit gate is applied on.

Your program will print the output quantum state, with one element at every line. Do not print anything other than the output quantum state. **The values should be printed with 3 decimal points precision, not more nor less. Print to stdio and not to a file!**

Ultimately, we will run your program like this,

```
./quamsimV1 input.txt > output.txt
```

And simply `diff` the output with the expected output. Therefore, make sure you are not printing anything other than the output quantum state.

We will test your program on the Hydra cluster, so make sure you get the expected outputs there. You will lose points if your code works on your personal computer, but not on the Hydra cluster.