FRQI Implementation: Qiskit vs. Cirq

Matt Harding | CMSC 657 | December 1, 2018

The Algorithm

For this assignment the algorithm I have implemented is the FRQI representation of quantum images. The algorithm transforms a state where all qubits are in the $|0\rangle$ state to the FRQI state, which stores an RGB image. Once the image has been stored the state is measured some large number of times to attempt to recover the RGB values in the image. There are three main phases for the algorithm. The first phase involves converting an array of RGB triplets, which represent an image, into angles in $[0, \frac{\pi}{2}]$. Each of these angles is then sent to a function which uses two operations to store these angles in a quantum state, keeping track of which angle represents a color in which pixel of the image. The algorithm starts with 2n + 1 qubits in the $|0\rangle$ state. The first operation applies Hadamard gates to 2n of the qubits. The second operation is a succession of rotations about the Y axis by $2\theta_i$, controlled by 2n qubits. The C-^{2n}-$R_y(2\theta_i)$ operations are broken down into CNOT and controlled y-rotation gates, to accomplish rotating by 2θ only the basis vector corresponding to the pixel whose color is being encoded in the state. Once all the colors have been encoded in this fashion, the state is final. The third phase of the algorithm is recovery of the image, which is done by repeatedly measuring the state to get estimates for the probability of measure $|0\rangle$ for each pixel and using that to recover θ_i. The θ values are then converted back to RGB values and the image is recovered. I have implemented this algorithm in Qiskit and in Cirq.

Language Comparison

In implementing the algorithm there were three main aspects where Qiskit and Cirq differed: the instantiation of the qubits, the creation of the circuit and the format of the results of running the circuit. In Qiskit the qubits are created as a group in a quantum register, with each one being referred to by an index in the register. This was fairly straight forward, but when printing the results of a measurement the indices for the qubits were the reverse of what I was expecting. The index 0 referred to the rightmost qubit, which I suppose makes sense in a binary counter sense of the qubits. When I first started using Cirq I used the "GridQubit" style of creating a lattice of qubits and referring to them by the location in the lattice. I then found out that you can use them individually and give each qubit its own name. I liked this because I could easily keep track of which qubit I was manipulating and measuring at different times in the algorithm. I realize there are advantages to using GridQubit, but since I am new to quantum programming I felt it was most important that the code was clear and readable.

In creating the quantum circuit the main difference I found between Qiskit and Cirq was the use of controlled gates. For Qiskit there is no controlled Y rotation gate, but there is a general controlled rotation gate. Using this gate and setting the φ and λ arguments to 0, turned this gate into a controlled Y rotation gate. Once I figured this out, it went smoothly. To get the controlled Y rotation using Cirq, I simply wrapped the Y rotation gate in "cirq.controlledGate()". This was very flexible and I could simply store this controlled gate in a variable and use it even more easily later on. To further simplify the process I created lists of controlled Y rotations for each angle and the reverse rotation of this angle all at once before adding them to the circuit. This way I could just refer to the index in these lists corresponding to

the pixel position, which I think makes the code more readable. This style of getting controlled gates is simple and intuitive, which could help on future programs as well.

When running the circuit Qiskit and Cirq are very similar, in that you can simply run the circuit and pass the number of "shots" or "repetitions" to the simulator. Where they differ is in how they format the results of running the circuit. Qiskit returns a very simple dictionary in which the keys are the basis vectors (stored as strings) and the values are the number of times the vector was the output. In order to use these results in the algorithm I cycled through every possible outcome and extracted the number of measurements from the dictionary. In Cirq the results of running the circuit are formed into a dictionary in which there is a key for each individual qubit. The values are lists, in which each element is a list of one element which is either "False" if the measurement for the qubit was 0 or "True" if the measurement was 1. I'm not sure why the individual measurements are in a list when there is only one element, but once I figured that out it was not difficult to deal with. To use the results given in this fashion I cycled through the lists for each qubit together, one measurement at a time to put together what the total outcome of each measurement was. This was one point for Qiskit, because it was much easier to interpret the results of a group of measurements simply by printing the output.

When learning to use a new language it is very important to have good documentation and be able to look at examples of using the language. I found the documentation for Qiskit very helpful, and there was a lot of discussion online about how to use different aspects of Qiskit. The documentation for Cirq was helpful for a few things, but was lacking in many others. For example, there was a section in the documentation about gates, but there was no gate given for a controlled rotation. After searching for a while, I did find a place where users were discussing controlled gates and I found out that there was function which would control any gate, which is what I used. Tutorials are also very useful when using a new language. While Cirq does have a tutorial, I found it much more technical than a beginning tutorial should be. It describes "variational quantum algorithms", "wave functions", and preparing an "ansatz state". I found this confusing and in strict contrast to the tutorial for Qiskit which included a "Hello World" section for quantum programming.

In comparing these two languages for this algorithm I decided to see if either language used more time to run the circuit with different measurements or if the results for accuracy of the image differed. To do this I added code to record the amount of time it took to run the simulation for both implementations for measuring between 100,000 shots and 1,000,000 shots in increments of 100,000. I also measured the average difference between the starting angles and the recovered angles for each of these runs. While the differences between accuracies did not seem significant, I did notice that Cirq consistently took more time to run the circuit, at times taking twice as much time as Qiskit. I would, of course, need to do more investigating before drawing any major conclusions, but I did find this interesting.

From everything I've learned about Qiskit and Cirq I would lean towards using Qiskit for future projects. From the tests that I did on the speed of the circuit running, it would seem to be faster than Cirq, of course I would need to do lots of further tests to confirm that. I also prefer the way that Qiskit presents the results of running a circuit. It is much easier to interpret at a glance than the way that Cirq does it. I also feel more confident that if I run into a problem with Qiskit that I will be able to find an answer in either the docs or discussion by others, although I imagine that Cirq's documentation will get better in time.