Michael Telahun

CECS 590 – Introduction to Quantum Computing

Dr. Sierra

Final Project

Grover’s Algorithm: Microsoft’s QSharp (Q#) vs IBM’s Qiskit

For my final project I chose to compare Grover’s search algorithm on a quantum simulator using the Microsoft’s QSharp (Q#) language vs IBM’s Qiskit language. Both are programming languages used for creating quantum circuits and algorithms. The initial goal was to find a set of gates or methodology that was implemented in one language and translate the code to the other. Then test the results of the experiments from the simulator. One of the several requirements for a quantum device is that the language. As operators are supposed to be agnostic of the device set it is paired with. To oversimplify, a circuit written for one device should have an equivalent implementation in a different device/language. This is because of the well-defined unitary and Hermitian quantum operators, such as Pauli X, Pauli Y, Pauli Z, Hadamard, and CNOT, which are mathematically general.

Grover’s algorithm is a search for querying data that provides quadratic speed up, , over the best case result achievable in a classical device, . It was initially intended for performing a database search. The solution to Grover’s search algorithm requires the use of an operator called Grover’s operator.

Where the oracle is represented as, O, in equation 1. The operator was proposed by Grover in 1998 and starts by finding the uniform superposition of each state. The oracle is in the form of equation 2

which finds the sum of each of the possible states for each of the superpositions [1,2]. This is encoded by the amplitude of a wave function where the general scheme can be thought of as solving an equation of the type where x is a specific amplitude or database value. When y is equal to 1 when the amplitude is found and 0 in all other cases.

For this project I was able to use the implementation of Grover’s search algorithm in [1] for IBM’s Qiskit language. Mandviwalla et. al provide implementations of Grover’s algorithm for two, three, and four qubits. The expansion of Grover’s algorithm requires an additional iteration of the Grover’s operator per each additional qubit after two qubits. There is a detailed description in [1] for the necessity of each of the gates in Table 1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Quantum Gate | Phase Gate | Qiskit Code | Q# Code | Comments |
| S, S† |  | qc.s(q[1]), qc.sdg(q[1]) | S(q1),  Adjoint S(q1) | Adjoint is equivalent to taking the Hermitian Conjugate |
| T, T† |  | qc.t(q[1]),  qc.tdg(q[1]) | T(q1),  Adjoint T(q1) | Adjoint is equivalent to taking the Hermitian Conjugate |
| CNOT |  | qc.cx(q[2],q[1]) | CX(q2,q1) |  |
| Hadamard |  | qc.h(q[1]) | H(q1) |  |
| Unitary |  | qc.u1(pi/8,q[1]), qc.u1(-pi/8,q[1]) | R(PauliZ, PI()/8, q1),  Adjoint R(PauliZ, PI()/8, q1) | Implementation from Qiskit and Q# varies internally for performing these operations but they are equivalent |
|  |  |  |  |  |
| Measure |  | qc.measure(q[1], c[1]) | M(q1) |  |

Table 1. Gates/Operators used for implementing Grover’s Algorithm in Qiskit and QSharp (Q#)

The first four rows of the table show gates which are generally defined as quantum operators meaning there should be an equivalent representation in Q#. The Unitary gate represents a general operator in a quantum device. Although the object definition in the language may vary from language to language the basis is a mathematical definition from literature. Table 1 shows the code for each of these gates in both Qiskit and Q#. The Unitary gate was the only gate which was not explicitly defined in both languages. It should be mentioned that each of the other gates is a specific implementation of the Unitary gates. A Unitary gate is a generalized operator or gate allowing for specific rotations in any of the locations within the Bloch sphere. The U1 gate in [1] is needed to perform the phase changes in the four-qubit expansion of Grover’s search. To perform these two operations in Q# I needed to understand exactly how Qiskit defines the unitary gates inside of the operator or function U1().

To start, the U1 gate implements the U3 gate. The U3 gate can perform rotations with respect to the Z axis in the Bloch sphere. The U3 gate takes as parameters These are angles or rotations which will affect the target qubit, and it is defined as a two-pulse single qubit gate. U1 gate only takes , which is also an angle or rotation, and it is defined in Qiskit as a diagonal single qubit gate. As the U1 gate only has one parameter the ( angles are set to zero. The matrix representation of the U3 gate is shown in equation 3.

The main difficulty I faced in this project was finding the correct translation of the U1 operator in Q# to perform the four-qubit expansion of Grover’s algorithm. The first two implementations for the two and three qubit circuits uses only the gates in the first four rows of Table 1. In Q# I found that the R gate can perform the same operation as U1. This is done by passing Pauli Z as the specified axis for rotation and the angle to rotate, see Table 1 for code. Once I was able to translate this operator, it was a matter of developing the code following the Microsoft documentation for Q#. The libraries mostly used were from Microsoft.Quantum.Intrinsic, .Arrays, and .Primitive. I was able to complete all the implementations for Grover’s algorithm in [1]. For each of the implementations I performed 8192 experiments as that is the largest number of shots available in IBMQ Experience. I believe this limitation exists as the simulator for IBMQ Experience is provided as a cloud service. Since Microsoft does not give access to a cloud device currently, simulations were run on my device for the Q# implementations.

Grover’s Algorithm Qiskit vs. Q# - 2 Qubits

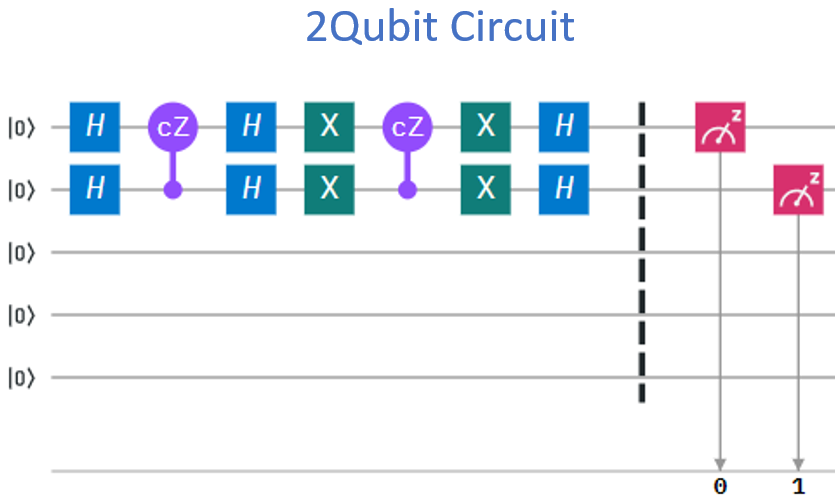


Figure 1. Two-qubit circuit drawing from IBMQ Experience.

The tests result for the experiments in both IBM and Microsoft simulators were the same, and I believe this is the expected behavior as the number of gates after decomposing the circuit in Qiskit is 18. The results are in Figure 2 for both Qiskit and Q#. Both simulations returned the sought state correctly 100% of the time in several of my experiments.

Figure 2. Qiskit and QSharp results for 2-Qubits: 8192 experiments.

Grover’s Algorithm Qiskit vs. Q# - 3 Qubits



Figure 3. Three-qubit circuit drawing from IBMQ Experience.

The test results for these experiments were very close see Figure 4. After decomposing the circuit in Qiskit there were a total of 33 gates. The best result I was able to achieve in the Qiskit implementation was 78.60%. The results for Q#’s were slightly higher with the highest score being 79.19%. Overall, I do not believe this difference of 0.59% means either simulator is better than the other, at least from my experiments.

Figure 4. Qiskit and QSharp results for 3-Qubits: 8192 experiments.

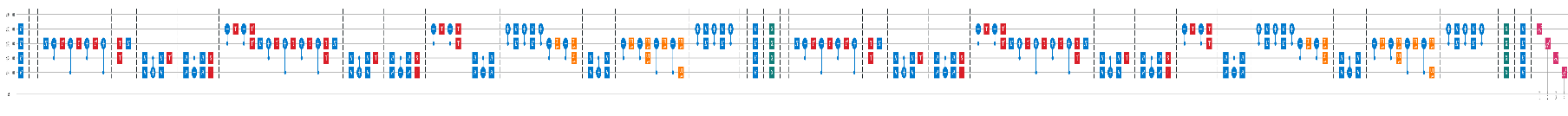
Grover’s Algorithm Qiskit vs. Q# - 4 Qubits

Figure 5. Four-qubit circuit drawing from IBMQ Experience.

The test results for these experiments were also very close but again slightly better in Q# see Figure 6. The number of gates after decomposing in Qiskit was 632. The best result I was able to achieve in the Qiskit implementation was 33.11%. The best results for Q# were 34.18%. The difference here is 1.07% and to the best of my knowledge does not prove either simulated devices/implementations are better than the other.

Figure 6. Qiskit and QSharp results for 4-Qubits: 8192 experiments.

After running these circuits several times and searching for different states, to the best of my knowledge neither device is superior to the other. In my experiments I did retrieve better results in Q# a handful of times but the results never varried more than 5% when compared to Qiskit. It was suprising for me to see that both backend systems were able to achieve nearly the exact same results. What was more interesting was that the behavior of the other qubits whose state we were not searching for, i.e. for the four-qubit implementation states 0000 through 1110. These states had almost the same distribution in both Qiskit and Q#. This detail I believe shows that the circuit conversion from Qiskit to Q# using the R(Pauli Z, PI/8) in place of the U1 gate was correct.

While I was searching for specific implmentations inside of Qiskit and Q# I noticed that the definitions for the actual gates in Qiskit were defined, specifically as a numpy array. This is in part why I was able to convert the four-qubit circuit to Q#. However, I was not able to find a definition for the actual gate operation inside of any function for Q#. It appears that interanlly the operators are not specified the way Qiskit has. For the conversion process I had to look at the documentation for Microsoft’s QSharp which gave matrix definitions for how operators are intended to work. Personally I found this a little interesting and discomforting as quantum computing is not a matter of just calling some function that retrieves a binary value. Quantum computing is a nondeterminanistic science and although we measure in terms of binary at the end the probablistic nature of the experiements requires as much insite as possible especially when developing. After reviewing much of the Q# code it was very interesting to see how many functions were implemented already in Q#. There are also very large notebooks in GitHub for Katas which are for learning how to use Q# and develop quantum circuits.

Overall from this project I learned a lot about the differences between Qiskit and Q#. I think more importantly I realized how hard it can be to generate a language around these nondetermanistic calculations in comparrision to their classical counter parts. I think the work that both IBM and Microsoft have done so far is incredible and that we are well under way to achieveing quantum supremacy in the next few years. I do not believe we have a clear winner today, at least from a simulator point of view, although IBM has several NISQ devices already available for public use. Personally I view QSharp more enjoyable to work with although Qiskit is probably more appropriate for the job.

References

[1] Mandviwalla, Aamir et al. "Implementing Grover’S Algorithm On The IBM Quantum Computers - IEEE Conference Publication". *Ieeexplore.Ieee.Org*, 2019, https://ieeexplore.ieee.org/abstract/document/8622457/.

[2] Coles, Patrick J. et al. "Quantum Algorithm Implementations For Beginners". *Arxiv.Org*, 2019, https://arxiv.org/abs/1804.03719.

Github Repository

<https://github.com/m0tela01/GroversAlgorithmQSvsQiskit>