Multi-Platform Implementation of Shor's Algorithm

Seong Cho

Department of Computer Science, Vanderbilt University

2201 West End Ave, Nashville TN 37235, USA

seong.g.cho@vanderbilt.edu

***Abstract*— This is a multi-platform implementation of Peter Shor's Algorithm of integer factoring with a five qubit circuit, with a view to factoring an integer using the same circuits. A QASM implementation using an online coding platform at quantum-inspire.com, a Python implementation with Qiskit library via Jupyter notebook through IBM Quantum Lab, and Q# implementation via Jupyter notebook and Microsoft's Quantum Development Kit at a local machine are achieved using the same principle. A slight change between the implementations is noted, and the results are assessed together.**

***Keywords*— Shor's Algorithm, Prime Factor, QASM, Qiskit, Q#**

I. Introduction

Shor's Algorithm was first introduced by Peter Shor in 1994 [1] as a computational algorithm for finding a prime factor of an integer. The factoring is done much more efficiently than previously known algorithms such as number field sieve [2], as instead of using an exponential polynomial as in the classical algorithm, it is achieving the factoring by finding a periodic modulo function and running it for multiple values in superposition to find the proper answers, and reducing the process into a simple polynomial.[3]

Actual implementations in the platforms are guided by Abijinth's[4]'s diagrams and instructions.

II. QASM Implementation

QASM implementation was achieved via Quantum Inspire. [5] It has the following code:

```
version 1.0

qubits 5

# start writing your code here

#Hadamard gate on 0, 1, 2
H q[0,1,2]

#entangle 2 and 3
CNOT q[2], q[3]

#entangle 2 and 4
CNOT q[2], q[4]

#Hadamard gate on 1
H q[1]

# controlled rotation on 0 and 1 by 1/2 phi
CR q[0], q[1], 1.57079632679

# Hadamard on 0
H q[0]

# controlled rotation on 1 and 2 by 1/4 phi
CR q[1], q[2], 0.78539816339

# controlled rotation on 0 and 2 by 1/2 phi

CR q[0], q[2], 1.57079632679

measure q[0:1,2]
```

And it shows the following diagram.

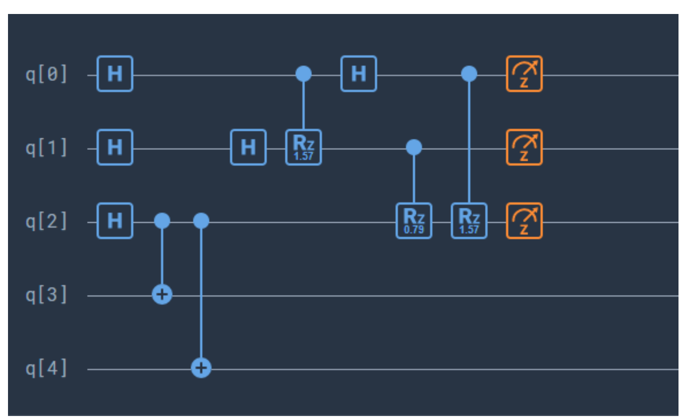


fig 1

The result from quantum inspire web simulation showed the following result. The simulation ran 4096 times over this circuit, and it showed that almost all results clustered over 2 values. This experiment was done on September 30, 2022.

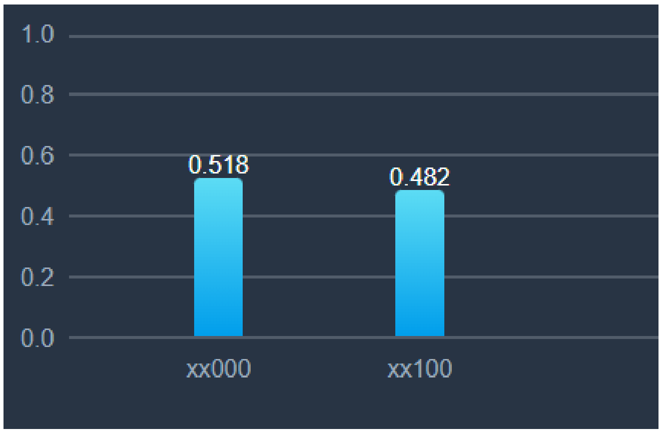


fig 2

As shown in fig. 2, the vast majority of the results show either |000> or |100> as the final reading of the collapsed wave functions.

III. QiskiT Implementation

The second implementation is done with QISKIT at IBM quantum lab [6]. QISKIT is a python library, and so the implementation was submitted with a Jupyter notebook. It required an IBM account at quantum lab.

The main part of the code is as follows.

```
qr = QuantumRegister(5, 'q')
cr = ClassicalRegister(4, 'c')
cir = QuantumCircuit(qr, cr)


cir.h(qr[0])
cir.h(qr[1])
cir.h(qr[2])

cir.cx(qr[3], qr[2])
cir.cx(qr[4], qr[2])
```

```
cir.h(qr[1])
cir.h(qr[0])

cir.rzz(np.pi / 2, qr[0], qr[1])
cir.rzz(np.pi / 4, qr[1], qr[2])
cir.rzz(np.pi / 2, qr[0], qr[2])

cir.measure(qr[0], cr[0])
cir.measure(qr[1], cr[1])
cir.measure(qr[2], cr[2])
```

Which resulted in a slightly different diagram from the QASM version, as the Hadamard gates and the controlled rotations have different orders.This resulting diagram and code was found by a few trials.

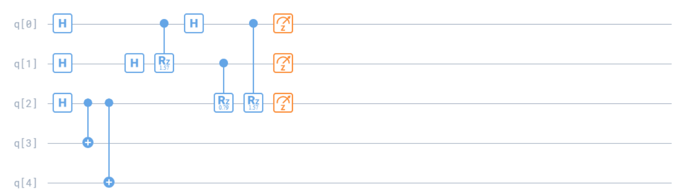
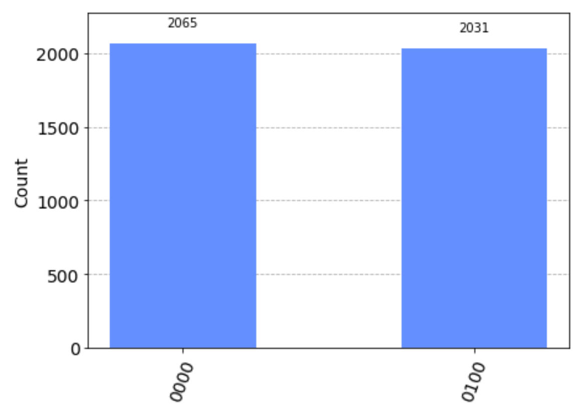


fig 3

It also showed that the vast majority of the results show either |000> or |100> as follows.



. fig 4

IV. Q# Implementation

The final implementation was done with Microsoft Quantum Development Kit with a local machine using Q#. The code shows similarity with the other implementations.

```
    operation RunShor(qs: Qubit[]): Int {

            //if the array of qubits has less than 5, it cannot proceed further. Return a default value.
        if (Length(qs) < 5) {
            return 0;
        }


        //add hadamard gates on first 3 qubits so that they would be phased.
        H(qs[0]);
        H(qs[1]);
        H(qs[2]);

        //entangle them with two control qubits
        CNOT(qs[2], qs[3]);
        CNOT(qs[2], qs[4]);


        //hadamard on 2nd qubit.
        H(qs[1]);

        //1/2 pi rotation with 1st and 2nd
        ControlledRotation(qs[0], qs[1], 1.57079632679);

        //hadamard on 1st qubit
        H(qs[0]);

        //1/4 pi rotation with 2nd and 3rd
        ControlledRotation(qs[1], qs[2], 0.78539816339);

        //1/2 pi rotation on 1st and 3red
        ControlledRotation(qs[0], qs[2], 1.57079632679);


        //measure and write out the binary results from first three qubits
        let outcome0 = (M(qs[0]) == One ? 0b1 | 0);
        let outcome1 = (M(qs[1]) == One ? 0b10 | 0);
        let outcome2 = (M(qs[2]) == One ? 0b100 | 0);

        //reset the qubits for next iteration
        for index in 0 .. Length(qs) - 1 {
            Reset(qs[index]);
        }

        return outcome0 + outcome1 + outcome2;
    }
```

The underlying circuit is exactly the same with the QASM diagram at fig 1. After running the circuit 2048 times at the local machine, the result was as follows.

```
Total: 2048
|>000 : 772
|>001 : 20
|>010 : 133
|>011 : 128
|>100 : 713
|>101 : 21
|>111 : 132
Errors : 130
```

fig 5

V. Discussion

While the three implementations result in more or less the same shape of circuits and similar results, there were still noticeable differences. The IBM QISKIT worked better with a slightly modified circuit.

Also the Q# result showed most noises and errors. It may be due to the environment. QASM and QISKIT ran remotely, while Q# ran locally.

References

[1] P. W. Shor, *Algorithm for quantum computation: discrete logarithms and factoring*, Proceedings 35th Annual .Symposium on Foundations of Computer Science, Santa Fe, USA, IEEE Computer Society Press, p 124. 1994.

[2] A.K.Lenstra, H. Lenstra, M.S. Manasse, J.M. Pollard, *Number Field Sieve*, Proceedings of twenty second annual ACM symposium on Theory of Computing, Tidmarsh College, Reading, Berkshire, UK. 1990

[3] D. Beckman, C. N. Amalavoyal, S. Devabhaktuni, J. Preskill, Op. cit.

[4] J. Abhijith., *Quantum Algorithm Implementations for Beginners*, arXiv e-prints, 2018

[5] https://www.quantum-inspire.com/

[6] https://qiskit.org/

[7] https://learn.microsoft.com/en-us/azure/quantum/overview-what-is-qsharp-and-qdk