Implementing Shor's Algorithm on various maps using IBM Q experience

Vijay Pawar
Seidenberg School of Computer Science and Information Systems
Pace University, New York
New York, USA
vp0395p@pace.edu

Abstract—Shor's Factorization Algorithm which was developed by Peter Shor is one of the most important algorithms in the field of Quantum Computing. We study the result of a version of Shor's algorithm on various qubit maps such as ibmq_qasm_simulator(up to 32 qubits) and ibmq_16_melbourne(15 qubits), for the particular cases of N = 9, 15, 21, 35. While implementing the Quantum Circuits for 7mod15 and 7^2mod15 on both of these simulators. The graphical representation of numbers is shown in this paper.

Keywords—Shor's Algorithm, Simulators, Quantum Circuits, Factorization

I. INTRODUCTION

Shor's Algorithm is a well-known example of quantum algorithm which outperforms the best classical algorithm. The factoring algorithm invented by Peter Shor depends on relation between the problem of factoring and order finding. The capacity of Shor's algorithm would be able to break through the various cryptography codes such as RSA public key cryptography system.

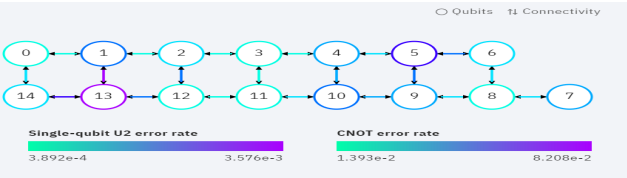
However implementing Shor's algorithm with physical Qubits is still a challenge because of the errors introduced due to an increase in the physical number of qubits and the Quantum gates needed to execute the algorithm. In this paper we implement the Shor's algorithm for N = 9, 15, 21, 35 on to two different qubit maps namely ibmq_qasm_simulator and ibmq_16_melbourne simulator in order to compare the results that are obtained. While implementing Shor's algorithm the number of physical qubits and circuit depth is kept to minimum in order to effectively reduce noise. The result is presented in the form of probability distribution of the values returned by the registers.

In this probability distribution the algorithm has to run many times in order allow the performance of quantum computers to be better evaluated. Probability plots are useful to visualize differences between probability distributions.

II. IBM Q EXPERIENCE SIMULATORS

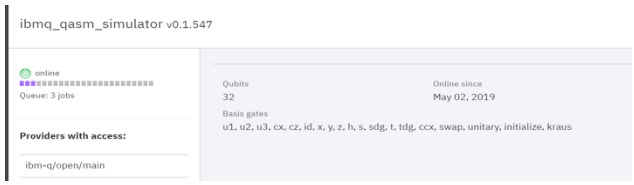
A. IBM MELBOURNE SIMULATOR (ibmq_16_melbourne)

The "IBM Q-16 Melbourne" simulator is a 15-qubit simulator with a connectivity that is provided by 22 CPW (Coplanar Waveguide) bus resonators each one connecting two quantum registers. For Melbourne simulator, every quantum register also has a dedicated CPW readout resonator attached for control. The below diagram shows the configuration and calibrations of ibmq_16_melbourne simulator.



B. IBM QASM SIMULATOR (ibmq_qasm_simulator)

The QASM simulator is the main Qiskit Aer backend. The backend emulates the execution of a quantum circuits on a real device and returns measurement counts. It includes extremely high configurable noise models and can be loaded with automatically generated approximate noise models which are based on the calibration parameters of actual hardware devices.



III. SHOR'S ALGORITHM

Shor's Algorithm is a quantum algorithm for factoring a number N in $O((\log N)^3)$ times.

The algorithm is important because it implies that public key cryptography may be easily broken, given a sufficiently large quantum computer. The RSA (Rivest–Shamir–Adleman) algorithm uses a public key N which is the product of two large prime numbers. One way to break RSA encryption is by factoring N, but with classical algorithms, factoring will become increasingly time consuming as N grows large. To be more specific, No classical algorithm is known that can factor in time $O((\log N)k)$ for any k. Shor's algorithm can break RSA in polynomial time. It has also been widely observed that Shor's algorithm attacks many more public key cryptosystems.

Like all Quantum computing algorithms, Shor's algorithm is probabilistic:

a. It gives the correct answer with high probability.

b. The probability of failure can be decreased by repeating the algorithm.

Shor's algorithm consists of two parts:

A. Classical Part

i. Pick a Random number a < N.

ii. Compute GCD(a, N). This may be done by Euclidean algorithm.

iii. If GCD(a, N) ! = 1 , then there is a non-trivial factor of N.

iv. Else, use the period-finding subroutine to find r, the period of the following function:

$f(x) = ax \bmod N$ where the smallest integer r is for $f(x + r) = f(x)$.

v. If r is odd, go back to Step 1.

vi. if a $r/2 \equiv -1 \pmod N$, go back to step 1.

vii. The factors of N are $GCD(ar/2 \pm 1, N)$.

B. Quantum Part

i. Start with a pair of input and output qubit register with log2N qubits each, and initialize them to

$N^{-1/2} \sum x|x\rangle|0\rangle$ where x runs from 0 to N - 1.

ii. Construct f(x) as a quantum function and apply it to the above state, to obtain

$N^{-1/2} \sum x|x\rangle|f(x)\rangle$

iii. Apply the Quantum Fourier transform on the input register. The Quantum Fourier Transform on N points is defined by:

$UQFT|x\rangle = N^{-1/2} \sum_y e^{2\pi ixy/N} |y\rangle$

This leaves us in the following state:

$N - 1 \sum_x \sum_y e^{2\pi ixy/N} |y\rangle|f(x)\rangle$

iv. We Perform a measurement obtaining some y in the input register and f(x0) in the output register. Since f is periodic, the probablity to measure some y is given by:

$N^{-1} | \Sigma x : f(x) = f(x0) e^{2\pi ixy/N|2} = N^{-1} | \sum_b e^{2\pi i(x0 + rb)y/N|2}$

v. Turning y/N into an irreducible fraction, and extract the denominator r', which can be a potential r.

vi. Check if f(x) = f(x + r').

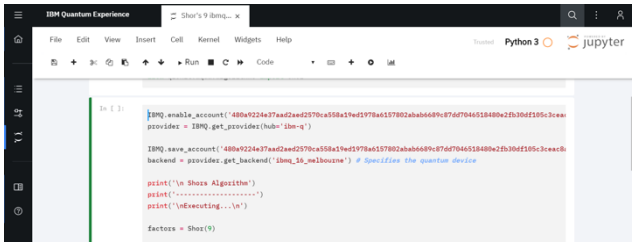
vii. Else obtain more potential values for r using values near y, or mutliples of r'.

viii. End

IV. IMPLEMENTATION

A. Implementing Shor's Algorithm for N = 9 on IBM_16_MELBOURNE

We start implementing Shor's algorithm on IBM Q experience using Qiskits. Using the (C) Copyright IBM 2019 (Shor) class for the algorithm we pass the N = 9 in ibmq_16_melbourne simulator. In order to calculate Prime factors for an integer n there has to be (2n+3) Qubits.




```
IBMQ.save_account('480a9224e37aad2aed2570ca558a19ed1978a6157802abab6689c87dd7046518480e2fb30df105c3ceac8
backend = provider.get_backend('ibmq_qasm_simulator') # Specifies the quantum device

print('\n Shors Algorithm')
print('--------------------')
print('\nExecuting...\n')

factors = Shor(21) #Function to run Shor's algorithm where 21 is the integer to be factored

result_dict = factors.run(QuantumInstance(backend, shots=1, skip_qobj_validation=False))
result = result_dict['factors'] # Get factors from results

print(result)
print('\nPress any key to close')
input()
```

After execution the result can be seen below in fig IV.4

```
 Shors Algorithm
--------------------

Executing...

[3]

Press any key to close
```

```
IBMQ.save_account('480a9224e37aad2aed2570ca558a19ed1978a6157802abab6689c87dd7046518480e2fb30df105c3ceac8
backend = provider.get_backend('ibmq_qasm_simulator') # Specifies the quantum device

print('\n Shors Algorithm')
print('--------------------')
print('\nExecuting...\n')

factors = Shor(35) #Function to run Shor's algorithm where 35 is the integer to be factored

result_dict = factors.run(QuantumInstance(backend, shots=1, skip_qobj_validation=False))
result = result_dict['factors'] # Get factors from results

print(result)
print('\nPress any key to close')
input()
```

B. Implementing Shor's Algorithm for N = 15, 21, 35 on IBMQ_QASM_SIMULATOR

When we implement Shor's algorithm for the integers N =15, 21, 35 an error is generated by the ibmq_16_melbourne simulator stating:

```
203     # Transpile circuits in parallel
204     circuits = parallel_map(_transpile_circuit, list(zip(circuits, transpile_configs)))

TranspilerError: 'Number of qubits (18) in circuit2 is greater than maximum (15) in the coupling_map'
```

This happens due to the fact that the number of Qubits required to factorize N = 15, 21, 35 is greater than the available qubits in ibmq_16_melbourne simulator that is 15 qubits.

Therefore, in order to execute the integers N = 15, 21, 35 we need to change the simulator to ibmq_qasm_simulator which has up to 32 qubits for execution.

```
In [ ]:
IBMQ.enable_account('d1b6f8a789a4echedeae4d4e88046dc2f82993a5a8dfa99b204126a6af80cd671c38aef4165e0b568e1
provider = IBMQ.get_provider(hub='ibm-q')

IBMQ.save_account('f21fd5c2f9d2e8f9e637992efb38c67ee82b5b9a34498748717bf5fae55527f3b34e16077b633063ad6c1
backend = provider.get_backend('ibmq_qasm_simulator') # Specifies the quantum device

print('\n Shors Algorithm')
print('--------------------')
print('\nExecuting...\n')

factors = Shor(15) #Function to run Shor's algorithm where 15 is the integer to be factored

result_dict = factors.run(QuantumInstance(backend, shots=1, skip_qobj_validation=False))
result = result_dict['factors'] # Get factors from results
```

Once executed it gives the following output for integers N = 15, 21, 35 respectively:

```
 Shors Algorithm
--------------------

Executing...

[[3, 5]]

Press any key to close
```

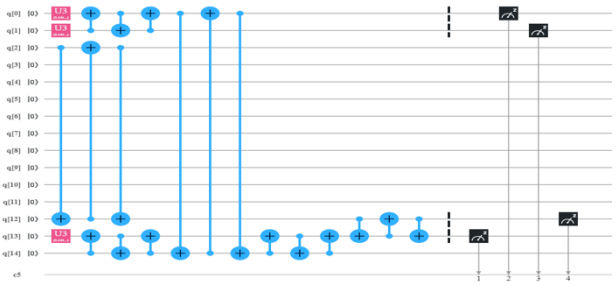
```
 Shors Algorithm
--------------------

Executing...

[[3, 7]]

Press any key to close
```

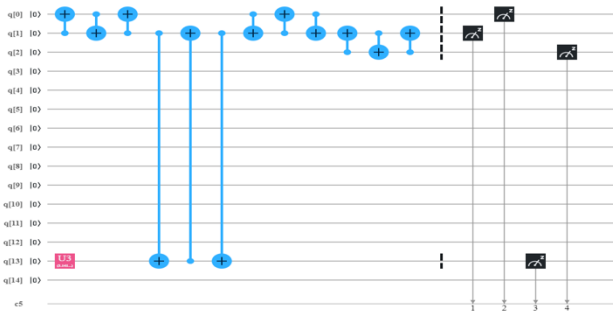
```
 Shors Algorithm
--------------------

Executing...

[[5, 7]]

Press any key to close
```

c. Executing 7mod15 and 7^2mod15 on Circuit Composer of IBM Q Experience

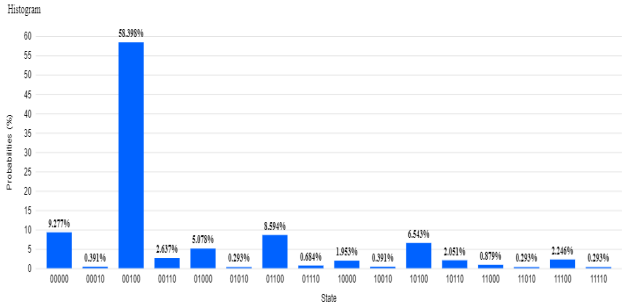
a) 7mod15 and 7^2mod15 on ibmq_16_melbourne simulator

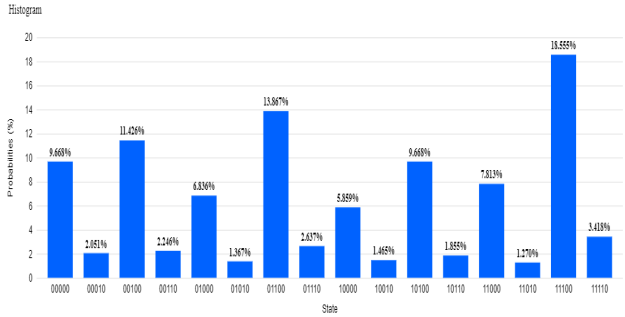
When we build circuits for 7mod15 and 7^2mod15 across five quantum states q|0>, q|1>, q|2>, q|3>, q|4>. When compiled on ibm_16_melbourne simulator the Transpiled circuit uses all 15 Qubits available on the simulator. The Transpiled circuits for 7mod15 and 7^2mod15 can be seen below respectively:





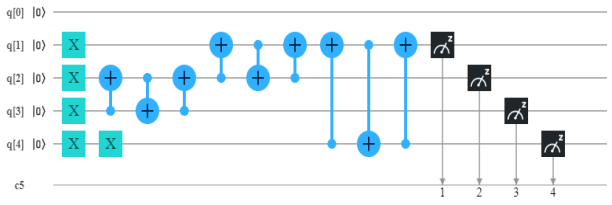
After execution the Probablity Distribution of the values returned by the simulators can be seen for 7mod15 and 7^2mod15 respectively:

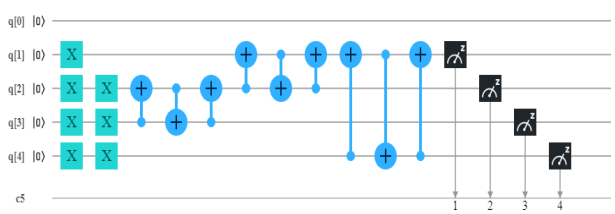




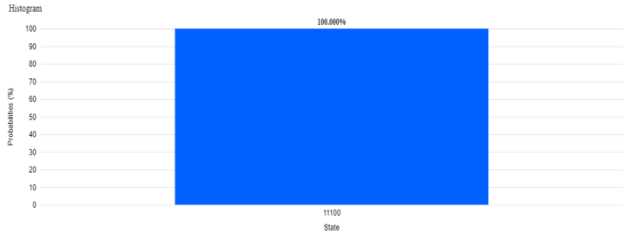
b) 7mod15 and 7^2mod15 on ibmq_qasm_simulator

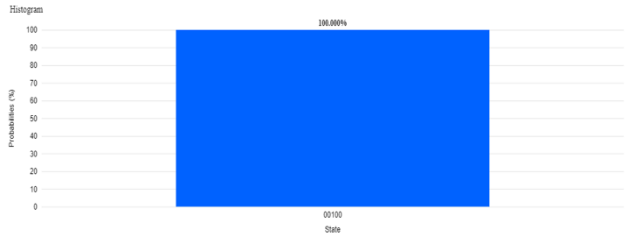
When we build circuits for 7mod15 and 7^2mod15 across five quantum states q|0>, q|1>, q|2>, q|3>, q|4>. When compiled on ibm_qasm_simulator, the Transpiled circuit uses 5 Qubits from the simulator. The Transpiled circuits for 7mod15 and 7^2mod15 can be seen below respectively:





After execution the Probablity Distribution of the values returned by the simulators can be seen for 7mod15 and 7^2mod15 respectively:



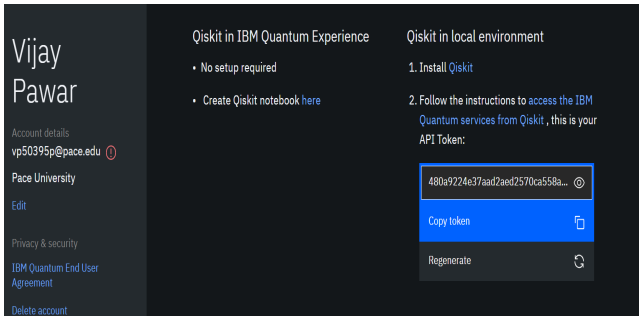


v. EXPERIENCES AND DATA ANALYSIS

A. IMPLEMENTING THE CODE

While implementing the code at first instance, we tried to implement the code for N = 21. At first, we need to create an IBM Q Experience ID in order to use the simulators. Once done so we need to generate an IBM API key which is needed to enable the account.

The below figure shows the generated IBM API key:



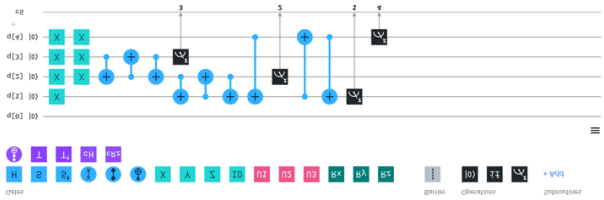
Next task is to implement the Shor's algorithm using the Shor's library which is (C) Copyright IBM 2019. When executing the Shor's python code, the simulator needs to be initialized so that we can use the qubits needed to factorize the numbers.

```
IBMQ.enable_account('480a9224e37aad2aed2570ca558a19ed1978a6157802abab6689c87dd7046518480e2fb30df105c3cea
provider = IBMQ.get_provider(hub='ibm-q')

IBMQ.save_account('480a9224e37aad2aed2570ca558a19ed1978a6157802abab6689c87dd7046518480e2fb30df105c3ceac8
backend = provider.get_backend('ibmq_qasm_simulator') # Specifies the quantum device
```

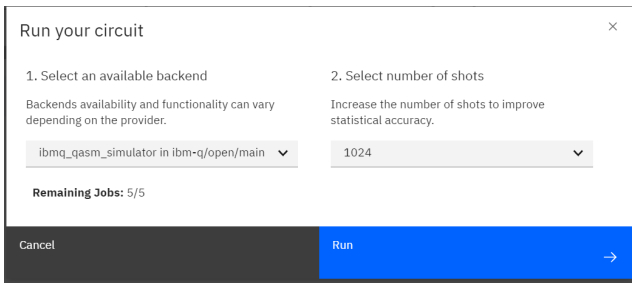
The major issue faced during implementation of Shor's algorithm was that the execution time taken by simulators would usually exceed and send the code into an execution error. Shor's algorithm needed to be implemented multiple times after which it would be executed once with no compilation error.

B. IMPLEMENTING THE CIRCUIT

While implementing the circuits in IBM Q experience, we need to use the circuit composer field in order to build the circuit first. Then we need to understand the use of registers to implement 7mod15 and 72mod15 as both the circuits would use different number of registers.



When the circuit is completed, we execute the circuit on both the simulators with an execution of 1024 shots so that the probability distribution can be most accurate.



REFERENCES

[1] P. Shor, in Proc. 35th Annu. Symp. on the Foundations of Computer Science, edited by S. Goldwasser (IEEE Computer Society Press, Los Alamitos, California, 1994), p. 124-134.

[2] Mirko Amico, Zain H. Saleem and Muir Kumph3, "An . Experimental Study of Shor's Factoring Algorithm on IBM Q".

[3] H.G Zhang, W.B Han, X.J Lai, et al,, "Survey on cyberspace security", SCIENCE CHINA, Information Science, vol. 58, no. 11, 2015, pp. 1-43.

[4] P.W Shor, "Algorithms for quantum computation: discrete logarithms and factoring", Proc. 35th Annual Symposium on Foundations of Computer Science, 1994, pp. 124-134.

[5] Micheal, A. Nielsen and Isaac, L. Chuang, "Quantum Computation and Quantum Information," Cambridge, Chapter 4, 2000.

[6] J. Proos and C. Zalka, "Shor's Discrete Logarithm Quantum Algorithm for Elliptic Curves," Quantum Info. Comput., vol. 3, no. 4, pp. 317–344, 2003.