Running Shor's Algorithm on IBM Quantum Experience

AP3421-PR Quantum Information Project (2017/18)

Submitted By:

Amitabh Yadav, Student No. 4715020
Pinakin Padalia, Student No. 4743644
Delft University of Technology

February 12, 2018

Abstract

A quantum algorithm is one that uses the quantum circuit model of computation for solving problems. It has been established that quantum algorithms can help accelerate the processing time of certain problems compared to their classical algorithm counterparts. One of the most famous example of such a quantum algorithm is Shor's Period Finding Algorithm for calculating prime factors of a number. Here, we present implementation of Shor's Algorithm on IBM-Q platform using QISKit, an open-source python based Software Development Kit (SDK). The algorithm is simulated on 'local_qasm_simulator' and implemented for IBM's 5-qubit hardware to demonstrate the factoring of the number 15. The results of the same is presented.

1 Introduction

Shor's Algorithm, originally published by Peter W. Shor in 1995 [1] for factoring large numbers is one of the quantum algorithms that substantially outperforms it's classical counterpart. The best known classical algorithm for finding factors of a number is the General Number Field Sieve (GNFS) and has a complexity of $O(2^{n^{1/2}})$ whereas Shor's Algorithm gives a polynomial-time execution for factorization. It is, strictly speaking, an algorithm for finding period of a function using quantum computer. The period finding is performed using quantum computing and is combined with classical procedures that help us to extract the factors of the number from the period of a chosen function.

The realisation of Shor's Algorithm on an actual quantum computer is a challenge due to the limitation on available quantum registers and the challenge of maintaining high fidelity control. However, several available quantum computing simulation and cloud-based hardware platforms have made it possible to design, simulate and run (certain) quantum algorithms. Examples of such simulation platforms include IBM-Q, Microsoft Liqui$|\rangle$, QuTech QX-Simulator and rigetti Forest. Among these, IBM-Q and rigetti Forest provide a developer environment for programming quantum circuits and implementing the circuit on a commercially available cloud-access based quantum computer.

The main focus of our project is on implementation of Shor's Algorithm for finding the factors of a number on the IBM-Q hardware platform. For demonstration, we picked a number that we want to factor N=15 and constructed the required unitaries and quantum circuit that performs the period finding operation. This quantum circuit was then integrated with classical algorithms such as for finding greatest common divisor (GCD) and developed as an application. The generalised circuit was first created and run on the local_qasm_simulator and then necessary changes were made to run on the IBM-Q 5-QuBit Quantum Computer.

This report presents the implementation methodology and results of running Shor's Algorithm on IBM-Q. Section 1 and 2 introduce the Shor's Algorithm and background work done for realisation of Shor's Algorithm. In section 3, we introduce the IBM-Q platform and the IBM QISKit, an open-source SDK for quantum programming. Section 4 presents the Shor's Factoring Algorithm. Next, in Section 5, we present the implementation methodology. Section 6 presents the results and Section 7 presents the conclusion.

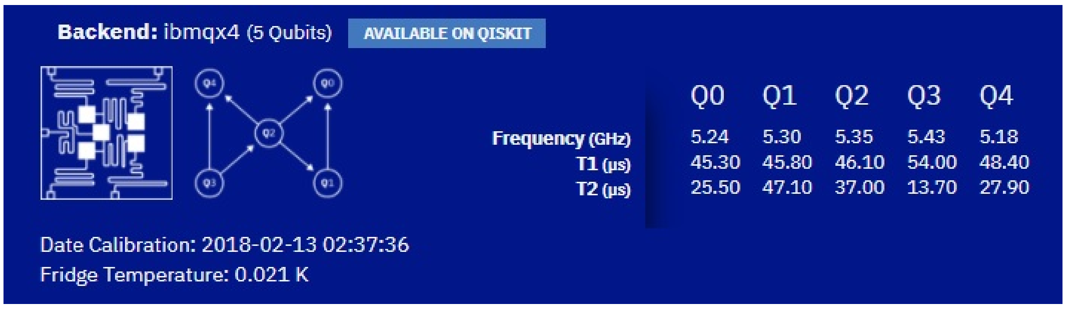
2 Background

The main agenda behind designing circuit for Shor's Algorithm is the modular exponentiation function for period finding. It is often denoted as a "black-box" operating on quantum register [2]. The circuit for Quantum Fourier Transform (QFT) is well established and can be implemented using Hadamard and Rotation gates [7][8][6]. However, factoring an n-bit number requires about $2n$ quantum registers for QFT and n quantum registers for modular exponentiation i.e. total $3n$ quantum registers.[2][5]. The availability of quantum registers is limited and does not allow factoring of very large numbers. However, Kitaev [3] showed that we need only the classical information from QFT (which is the period r) and we can replace the $2n$ qubits of the QFT register by a single qubit. In another implementation, Shor's Algorithm for factoring has been implemented using $2n + 3$ qubits. [5]

3 IBM Quantum Experience

IBM Quantum Experience is an IBM-Cloud based platform that enables anyone to connect to one of the IBM's Quantum Processors kept at Thomas J. Watson Research Center in New York, USA. It is an initiative that have made universal quantum computers commercially available for scientific collaboration.

IBM-Q was launched in May 2016, initially with a 5 QuBit Quantum Processor and a simulator and by 2017, they added a 16-QuBit Processor[1] and 20-QuBit Processor[2]. The IBM-Q 5-QuBit Processors ibmqx2 and ibmqx4 are available to run algorithms. The details of ibmqx4 (Raven) Quantum Chip is shown in figure 1. Initially the IBM-Q platform was designed such that users can interract with the processor by making quantum circuit on the online GUI based Quantum Composer and/or by writing Quantum Assembly (QASM) Code. Later, IBM has made available a Python API and SDK (called QISKit) that allows to simulate and interact with the Quantum Processors.



Backend: ibmqx4 (5 Qubits) AVAILABLE ON QISKIT

	Q0	Q1	Q2	Q3	Q4
Frequency (GHz)	5.24	5.30	5.35	5.43	5.18
T1 (µs)	45.30	45.80	46.10	54.00	48.40
T2 (µs)	25.50	47.10	37.00	13.70	27.90

Date Calibration: 2018-02-13 02:37:36
Fridge Temperature: 0.021 K

Figure 1: IBMQX4 Raven 5-QuBit Quantum Chip Specifications

The Quantum Information Software Kit (QISKit) is a software development kit that allows to create, compile, simulate and run quantum circuits in real time on the IBM-Q backend platforms. It uses the Python API to connect to the IBM-Q Processors and is developed for Python 3.5. QISKit can be installed directly using pip install and we can write quantum circuit scripts that can be run either on the local_qasm_simulator or on a real Quantum Chip.

[1]under maintenance
[2]available only to IBM-Q partners

4 Quantum Factoring Algorithm

The quantum factoring algorithm works by finding the period of a (carefully chosen) function by quantum methods and using classical means to extract the factors of the number from its results.

Let us assume a number that we want to factor N=15. If N is even, we return the factor 2. Next we check classically, if N is of the form p^k. If true, we return k as the factor. If the above cases fail, we proceed ahead and select any random number, say $a \in [2, N-1]$. We call it, the base. Next, we calculate the GCD, $t = gcd(a, N)$. If $t > 1$, then a and N are not co-primes; hence t is a factor. For example, for N=15 and $a = \{3, 5, 6, 9, 12\}$, $t > 1$ therefore, t is a factor. However, for cases of $a = \{2, 4, 7, 8, 10, 11, 13, 14\}, t = 1$ and therefore, we proceed ahead. The next step is finding the period of the function $r = a^x modN$. The task of finding period of the function is not trivial and hence requires quantum implementation. If r is odd or $a^{r/2} + 1 = 0modN$, then nothing can be concluded and therefore we go back and choose another random number a and repeat the above steps. It is observed that this happens at most with 50% probability. However, if r is even and $a^{r/2} + 1 \neq 0modN$ then $gcd(a^{r/2} \pm 1, N)$ is a factor. For example, if N=15 and $a = 7$, the modular exponentiation function gives us $r = 4$; therefore $gcd(7^{4/2} \pm 1, 15) = \{5, 3\}$ is factor.

To implement the factoring algorithm on the Quantum Computer we need to implement $a^x modN$ modular exponentiation function for $x = 1, 2, 3...$ in the computational register and then retrieve the period, r. For this, the Quantum Fourier Transform (QFT) is applied in the period registers and the period is extracted by performing quantum measurements.

5 Methodology

A block diagram of the Quantum Circuit for finding factors of a number is given in figure 2. It has two QFT and one modular exponentiation function. The circuit for QFT can be constructed using Hadamard gates, controlled Rotation gates and Swap.

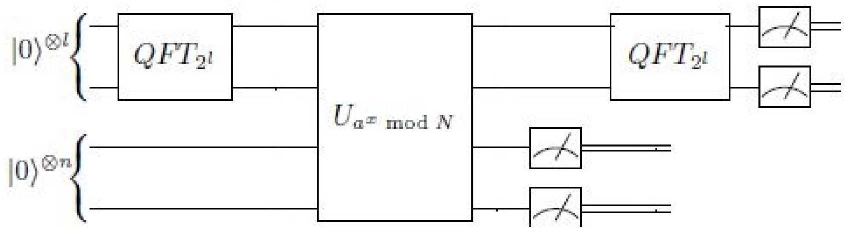


Figure 2: Block Diagram for Shor's Period Finding

It is however difficult to implement Shor's Algorithm on Quantum hardware due to limitation on the size of the quantum registers. To factor a number N, an $n = \lceil log_2(N) \rceil$ bit number, we need at least n computational qubits and generally about $l = 2n$ qubits in the period register. Therefore, in total about $3n$ qubit registers are required. The implementation of Shor's Algorithm presented in the report is based on the article 'Realisation of Scalable Shor Algorithm' published in Science Magazine [2].

We use Kitaev's [3] approach to finding QFT which helps to replace the QFT by a single qubit. This is done by single qubit recycling (qubit state readout and reinitialization). This reduces the number of qubit registers required to run the quantum algorithm from $3n$ to $n + 1$ qubit registers. This implementation is shown in figure 5B

The second main component of Shor's period finding is implementing the modular exponentiation function. This function is implemented by using modular multipliers. We can write the modular exponentiation function as:

$$a^x modN = (a^{2^0} modN)^{x_0} \times (a_{2^1} modN)^{x_1} \times ... \times (a^{2^{2n1}} modN)^{x_{2n1}} modN$$

where, x_i are the bits of binary expansion of x. The circuit implementation of this method is shown in figure 3.[4] Here, a modular multiplication $|a^y modN\rangle$ is performed on each of the multipliers in the computational register when the control qubit is $|1\rangle$. This circuit implementation returns the output $y.a^x modN$ on the computational register. Moreover, in the single qubit QFT implementation we see that, all previous measurements of the inverse QFT determine the successive phase shifts on the period register.

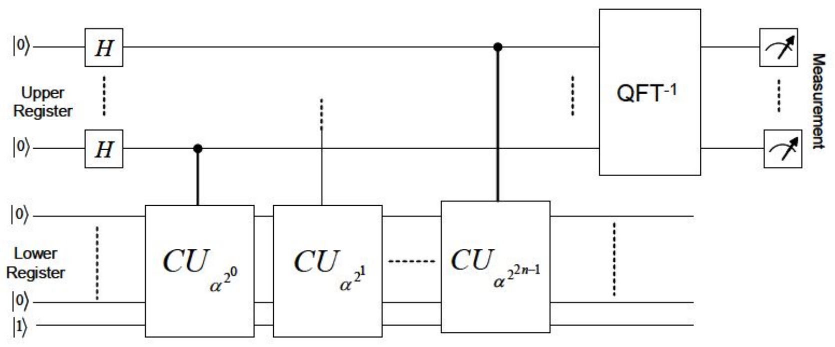


Figure 3: Design of Modular Exponentiation using controlled Modular Multipliers [4]

The generalised simulated circuit for period finding is as shown in figure 5D and 5E; and circuit implemented on IBM-Q 5-Qubit Quantum Chip is shown in Figure 5C. Here, the same approach to find the modular exponentiation is implemented using controlled modular multiplication. Another important aspect of this implementation is that it uses "qubit-recycling" viz. regenerating a single qubit again and using feed-forwarding. This is according to the

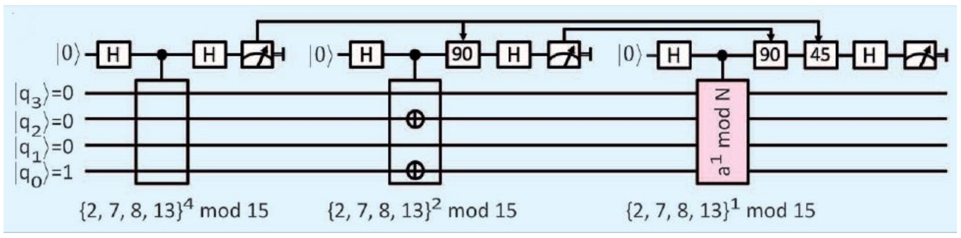


Figure 4: Design of Modular Exponentiation Circuit using single qubit [2]

Kitaev.[3] This allows using only a single qubit in the period register and thus allows the implementation of Shor's Algorithm on the IBM 5-Qubit Chip, that otherwise requires a minimum of $3n = 12$ qubits.

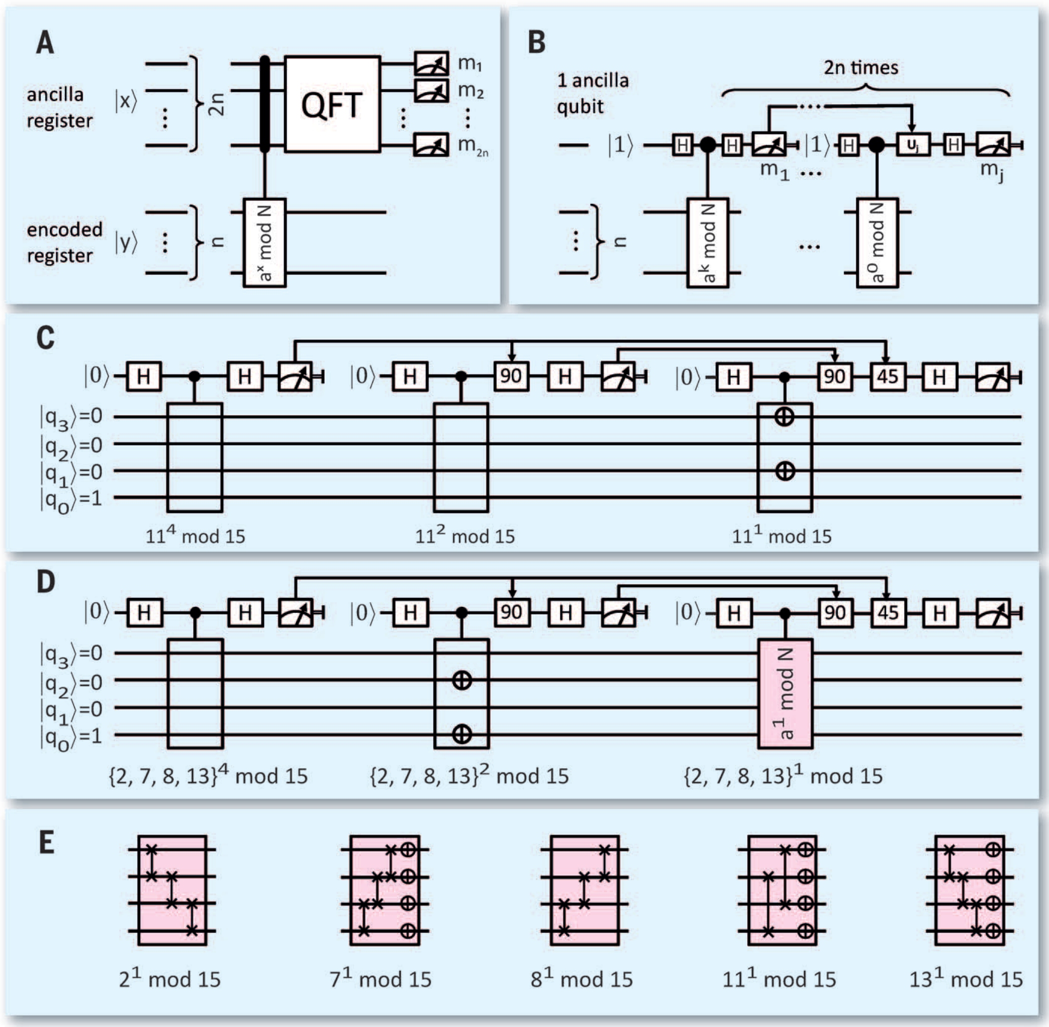


Figure 5: Quantum Circuit for Period Finding [2]

A description of the algorithm followed for generalised factorising is as follows:

Algorithm 1 Shor's Algorithm for factoring N=15

```
1: procedure FACTOR(N)
2:     $n = \lceil log_2(N) \rceil$
3:     If (N is even) return 2
4:     end If
5:     If $N = p^k$ form return p is a factor
6:     Choose base $a \in \{2, ..., N-1\}$    ▷ a=11 for hardware implementation
7:     $t = gcd(a, N)$
8:     if $t > 1$ return $t$
9:     end If
10:    $r = period(x \mapsto a^x modN)$
11:    If r is odd then go to 6
12:    end If
13:    return $gcd(a^{r/2} \pm 1, N)$    ▷ Factors are returned here
```

The algorithm for period finding using Modular Exponentiation is discussed as follows:

Algorithm 2 Modular Exponentiation Function for Period Finding

```
1: procedure PERIOD($a^x modN$)
2:     create QuantumProgram() Object
3:     create QuantumRegisters with 5 qubits
4:     create Classical Registers with 3 qubits    ▷ Measurement registers
5:     create QuantumCircuit and get circuits and registers by name
6:     perform modular exponentiation by modular multiplication
7:             create unitary $a^4 mod 15$
8:             create unitary $a^2 mod 15$
9:             create unitary $a^1 mod 15$
10:    feed forward and measure the classical register    ▷ Qubit Recycling
11: while True do    ▷ Run continuously until r is even && $a^{r/2} + 1 \neq 0$
12:    execute QuantumCircuit
13:    get measured valued from classical register, b
14:    k = gcd($b/2^l$)
15:    r = $2^l/k$
16:    If r is even and $((a^{r/2} + 1) modN \neq 0)$ then break
17: return $r$    ▷ The period is r
```

6 Implementation

Design Space Exploration After careful consideration of the problem statement and discussion about the possible implementation of Shor's Algorithm, we listed the following three implementations of the project:

1. 12-Qubit Simulation - Generalised (without Qubit Recycling) (on local_qasm_simulator)
2. 5-Qubit Simulation - Generalised (on local_qasm_simulator)
3. 5-Qubit Hardware - Generalised and particular case, a = 11 (on ibmqx4)

Implementation For implementation of Shor's Algorithm, we need to have QISKit along with python 3 and all the required dependencies installed. The details of this can be found in [7] [8]. The first steps were to define the classical components of the Shor's Algorithm which include I/O, *mod*2 and *gcd* functions.

For 12-Qubit Generalised Simulation we chose to implement the circuit given in figure 3. The circuit is give in figure 5E. The final result of this implementation did not fetch us promising results due to being stuck in an iterative loop thus not getting an output.

The aim of the project is running the Shor's Algorithm on IBM Quantum Hardware. So, we need to implement the quantum circuit for IBM-Q 5-Qubit Quantum Chip. So, the next implementation was using Qubit Recycling to eliminate to 2n period registers and using only 1 qubit in it's place. We could thus use only 5-Qubits in the python script. We used the reference from [2] to design the quantum unitaries. The circuit implementation for generalised 5-Qubit simulation is given in figure 5D and 5E. We ran the same circuit on the local_qasm_simulator and the circuit executes without any errors. The code for 5-Qubit Shor's Algorithm generalised simulation is on GitHub.

The final step is the implementation on 5-Qubit Quantum Hardware. Running the simulation code for quantum hardware requires specification of the coupling map of the ibmqx4 quantum chip and specifying the appropriate backend. However, after examining the coupling map of ibmqx4, we found that a generalised implementation of Shor's Algorithm on the chip is not possible due to unavailability of coupling between 3 qubits with a center control qubit. The same is given in figure 6. Here, we see that Q2 can control operations between (Q4, Q3) and (Q1, Q0). But , if we see figure 5E, we find that in all unitaries except $11^1 mod 15$ we need a swap operation between 3 pairs of Qubits. This coupling map however, does not allow this. So, we conclude that we only run Shor's Algorithm for a particular case of a=11. The code of the 5-Qubit implementation of Shor's Algorithm for a particular case of a=11 is on GitHub.

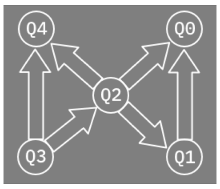


Figure 6: Coupling map ibmqx4

Challenges The main challenges faced during the completion was in the initial stages of Installation of QISKit. The QISKit was developed for an older version of Scipy (0.19.1) and did not install properly on our systems. This problem persisted for weeks and we informed about this to IBM-Q team through slack community and later when IBM-Q released a new version of QISKit compatible with Scipy 1.0.0, only then we were able to begin work.

The second challenge is the coupling map of ibmqx4 which does not allows for implementation of the generalised Shor's Algorithm of a 4-Bit number. We can only run the Shor's Algorithm for a particular case of a=11 on the 5-Qubit hardware. However, the generalised 5-Qubit Shor's Algorithm works on simulation without any errors.

7 Results

By running Shor's Algorithm for 5-Qubit on local_qasm_simulator we were able to infer a number of results. We fixed the number to be factorised N=15 and used a random function to generate the values of a. The program then proceeds according to Algorithm 1.

In order to formulate various statistics, we ran the same algorithm in loop for 100 times, saving the results and necessary intermediate values in a an .xls file.

1. Each time for a randomly generated value of a, we cannot predict if we can get factors of a number. In this case, we can get a graph of the cases when the algorithm was able to used the Quantum Period Finding (QPF) and when the value of a did not need to use QPF. See figure 7.

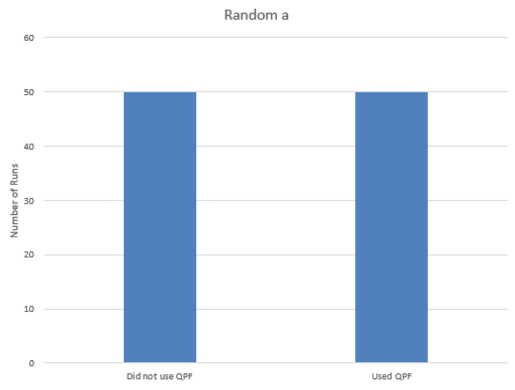


Figure 7: For any random a, probability of running QPF

2. Now, we fixed the value of a to different cases when QPF would run viz. a={2,4,7,8,11,13,14} and noted down the number of runs for which the QPF gave the corrent period in 1 run (blue), 2 runs(red), 3 runs (green)

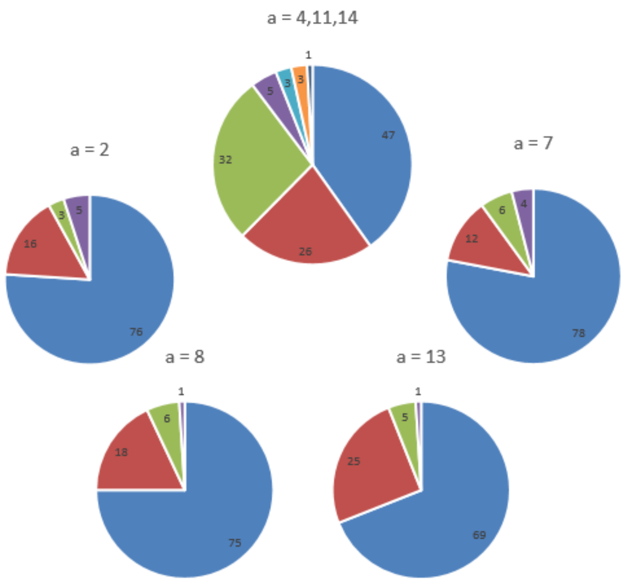


Figure 8: Number of QPF runs for different values of a

and so on. Figure 8 shows the number of QPF runs for different values of a to obtain the period.

3. Using the above result of obtaining the period of the function for different values of a, we can infer the result of when the algorithm would take only 1 run to give the correct period, 2 runs, 3 runs and so on. This is shown in figure 9

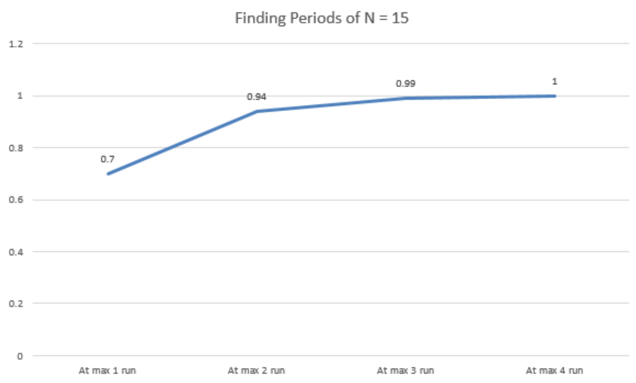


Figure 9: For any random a, probability of running QPF

4. Finally, we ran the code for 5-Qubit Shor's Algorithm on IBM-Q ibmqx4 Quantum Chip for 1000 runs. This can be run for the case N=15 and a=11. In this case the QPF function returns the period $r = 2$. The pie-chart in figure 10 shows the number of cases when QPF function returned and when it did not return the correct period.

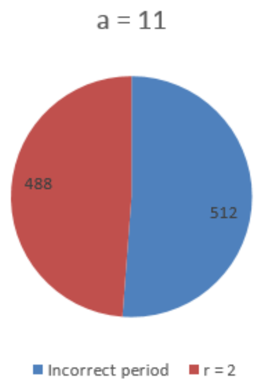


Figure 10: For any random a, probability of running QPF

8 Conclusion

This report documents the project development of 'Shor's Algorithm implementation on IBM-Quantum Experience'. The main part of implementing Shor's Algorithm on Quantum Hardware is the Quantum Period Finding. A number of different approaches to implement Shor's Algorithm circuit are present [2] [5]. However, for implementing circuit on ibmqx4 hardware with 5 Qubits we required an algorithm that utilises only 5 qubit. We followed the approach of [2] and implemented Shor's Algorithm for factoring N=15 by using 4 qubits as computational qubits and 1 qubit as period qubit. We conclude the following:

1. Shor's Algorithm runs on IBM-Q ibmqx4 hardware platform for N=15 and a=11; and returns the correct factors.

2. The generalised version of Shor's Algorithm runs on IBM-Q local_qasm_simulator for N=15 and any random value of a; and returns the correct factors.

3. The IBM-Q hardware is capable of feed-forwarding (Qubit Recycling) is not confirmed. Since in the 5-Qubit hardware implementation of Shor's Algorithm, the first two unitaries of $11^4 mod 15$ and $11^2 mod 15$ are basically Identities; so, even after implementing feed-forwarding in the algorithm, no actual feed-forwarding is happening on the hardware.

4. There are different hardware challenges to implementing a generalised case of Shor's Algorithm for period finding, such as,
   a) Scalability: More number of Qubits are required to run Shor's Algorithm as the value of N grows.
   b) Generality: The coupling map of ibmqx4 does not allow the implementation of generalised Shor's Algorithm for any random a on the hardware.

References

[1] Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. `arXiv:quant-ph/9508027`

[2] Monz, T., Nigg, D., Martinez, E.A., Brandl, M.F., Schindler, P., Rines, R., Wang, S.X., Chuang, I.L. and Blatt, R., 2016. Realization of a scalable Shor algorithm. Science, 351(6277), pp.1068-1070.

[3] A. Y. Kitaev, http://arxiv.org/abs/quant-ph/9511026 (1995).

[4] Fast Quantum Modular Exponentiation Architecture for Shor's Factorization Algorithm. arXiv:1207.0511

[5] Circuit for Shor's algorithm using 2n+3 qubits. `arXiv:quant-ph/0205095`

[6] IBM Quantum Experience Documentation/Full User Guide. https://quantumexperience.ng.bluemix.net/qx/tutorial?sectionId=full-user-guide&page=introduction

[7] https://github.com/QISKit/qiskit-tutorial

[8] https://github.com/QISKit/qiskit-sdk-py