Performance of Shor's And Grover's Algorithm on a Scalable Quantum Computer

By

Yousef Sadrossadat

Supervisor: Glenn Gulak

April 2018

Performance of Shor's And Grover's Algorithm on a Scalable Quantum Computer

By

Yousef Sadrossadat

Supervisor: Glenn Gulak

April 2018

Abstract

The development of general purpose quantum computers has experienced a massive growth in recent years as result of investments from tech industry leaders such as Google and IBM. As such, the number of qubits in quantum computers has significantly increased. The imminent arrival of quantum computers poses a threat to public key cryptography algorithms based on integer factorization, discrete logs and elliptic curve. Shor's and Grover's are two quantum algorithms that cause security concerns. It is critical to understand the resource complexity of these algorithms on a quantum computer, in order to prevent security attacks and move to quantum safe algorithms in time. Experimental realizations of Shor's algorithm utilize compiled circuits for modular exponentiations, which is currently the bottleneck of the algorithm. Simulation results for Shor's subroutine indicate that $O(10^3)$ qubits and $O(10^9)$ gates are required to break RSA key size of 2048 bits. The current state of quantum architecture is capable of $O(10^3)$ gates with $O(10^2)$ qubits. The computation power of quantum computers does not solely rely on the number of quantum qubits. Moreover, depth of realizable circuits, connectivity of the qubits and number of parallel operations are key factors to consider for computational power.

Acknowledgements

I would like to thank my supervisor Prof. Gulak for his continuous support and guidance. His vision and encouragement has always kept me motivated.

Table of Contents

List of Figures

List of Tables

List of Symbols

a	Constant used in modular exponential function
α	Complex number denoting linear combination coefficient
β	Complex number denoting linear combination coefficient
I	Identity matrix
μ	Mean
N	Integer to be factored
p	Prime factor
q	Prime factor
r	Period of the modular exponential function
x^*	Target element under query in database
$\|\Psi\rangle$	Arbitrary quantum state as column vector known as ket
$\langle\Psi\|$	Arbitrary quantum state as row vector known as bra
†	Conjugate transpose
$\|+\rangle$	Super position state $H\|0\rangle$
$\|-\rangle$	Super position state $H\|1\rangle$

List of Function

CU	Controlled phase gate
FFT	Fast Fourier transform
GCD	Greatest common divisor
H	Hadamard Gate
O_f	Oracle function
U	Unitary matrix which signifies an arbitrary quantum operation
U_d	Grover diffusion gate
X	X Pauli gate with rotation of π around x axis
Z	Z Pauli gate with rotation of π around z axis
Z_C	Controlled Z Pauli gate
Y	Y Pauli gate with rotation of π around y axis
QFT	Quantum Fourier transform

1. Introduction

1.1 Context

In recent years, research on the development of large-scale quantum computers has accelerated at an exponential pace, with tech giants Google, Microsoft and IBM each investing to accelerate the construction of the first commercial general-purpose programmable quantum computer [1]. Preliminary results from recently developed, small-scale quantum prototypes (up to 17 qubits) such as the IBM's Quantum Experience (accessible via IBM Cloud), demonstrate fidelity and controllability of the hardware [2]. Such prototypes hint at the imminent arrival of this technology as systems of O (100) qubits are within reach and the challenge ahead is to increase the computational power of these processors [3]. Quantum devices (with more than ~50 qubits) are expected vvvvvvvvto perform certain algorithms beyond the capabilities of the today's best classical supercomputers known as Quantum Supremacy [4].

The implication of this fact is that general-purpose quantum computers will significantly weaken the security of many commercially important cryptographic algorithms. Classical computers can efficiently multiply prime numbers, however factoring the product of prime numbers requires enormous computational effort. This asymmetry of computational effort forms the basis for public-key encryption [5]. Peter Shor introduced an efficient quantum algorithm in 1994 for factoring large integers into its prime factors in polynomial time [6]. The speedup in quantum algorithms such as Shor's and Grover's will render some important public-key and symmetric cryptography algorithms such as RSA and AES inadequate [7] [8].

1.2 Objective

The goal of this paper is to develop test benches for Shor's and Grover's algorithm in order to gain insight on their transformative impact on cryptography algorithms. Small scale circuits will be designed for execution on simulation platforms and available hardware processors. The simulation engines will include Project Q and IBM Quantum Experience. Quantum hardware will be accessed through IBM Quantum Experience cloud service to test IBM QX5, their 16-qubit processor. The small-scale models will be used to analyze the complexity of each algorithm on a quantum computer and benchmark the current computation power of these machines. Furthermore, the

results will be used to identify requirements for a general-purpose quantum computer to execute Shor's algorithm and estimate the resources required to break RSA keys.

The remainder of this report will focus on a review of quantum computation, followed by survey of the current state of quantum hardware architecture. These sections will be proceeded by detailed examination of each of the Grover's and Shor's algorithms.

2. Quantum Domain

In order to analyze the performance and examine the algorithms in detail, the subsequent sections will present an introduction to quantum computation and the current state of quantum hardware architectures.

2.1 Quantum Bits

In a classical computer, a bit of information is produced by setting the voltage over wire to either low or high which corresponds to a value of 0 or 1. Similarly in a quantum system, a quantum bit known as a qubit, has two common states $|0\rangle$ and $|1\rangle$. The qubit states are represented using Dirac Bra-Ket ($|\rangle$) notation, in order to easily distinguish between classical and quantum bits and enable easier illustration of qubit operations. Contrary to a classical bit, each qubit state is characterized using two complex numbers, forming a 2d vector in $\mathbb{C}^2$. The two common states $|0\rangle$ and $|1\rangle$ have the following vectors $|0\rangle=\begin{bmatrix}1\\0\end{bmatrix}$, $|1\rangle=\begin{bmatrix}0\\1\end{bmatrix}$, known as the standard basis. Furthermore, a qubit does not necessarily have to be in only one of the common state $|0\rangle$ and $|1\rangle$, instead it can take superposition of the two states. Superposition is a linear combination of the standard basis vectors. An arbitrary state $|\Psi\rangle$ (a 2D complex vector) can have the following composition $|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$. The coefficients α and β are complex values known as the amplitudes with the normalization constraint $|\alpha|^2 + |\beta|^2 = 1$. The Uncertainty Principle shows that any measurement will only provide partial information about the state of the qubits and it will also inevitably disturb it. Accordingly, the values of α and β cannot be directly measured. Instead, much like a classical system, where the evaluation of single bit returns a value of either 1 or 0, measurement of a qubit state $|\Psi\rangle$ gives the result of either 1 or 0. The major difference is that the quantum measurement is probabilistic, and in this case, it will return the value 1 with probability $|\beta|^2$ or the value 0 with probability $|\alpha|^2$. Interestingly, measurement of a qubit in superposition, will collapse the state into one of its basis vectors with a probability proportional to the superposition coefficients.

2.2 Decoherence

A critical requirement for implementation of a general-purpose quantum computer is the ability to preserve quantum coherence. Coherence is a fundamental property of quantum mechanics, where different states have fixed phase relations. The challenge is that the quantum systems cannot be

perfectly isolated and through interaction with their environment they will lose coherence, which will lead to loss of information. In general, there are two decoherence processes, energy relaxation and dephasing, as are illustrated in the figure 1. Energy relaxation is the loss of energy of the qubit over time where for example the excited state $|1\rangle$ decays to $|0\rangle$ as the qubit interacts with its surroundings. Dephasing affects superposition states due to disturbances in the environment, where the mutual phase between qubits is destroyed. Both of these processes have time constants (T1 for energy relaxation and T2 dephasing) that are regularly measured and compared as benchmarks for various quantum system. The longer the time constants, the lower the error rates and consequently leads to more computation time. The coherence times have been significantly improving over the past decade. Figure 2 depicts the improvements of dephasing time constant by a factor of 10 for every two years on IBM machines.

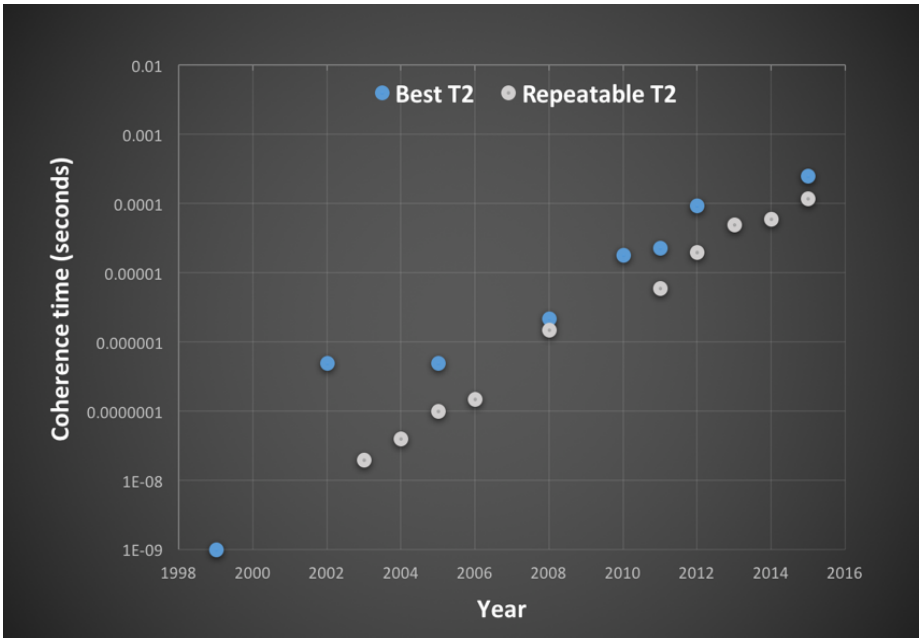

Figure 1: Dephasing (T2) vs. Time [27]

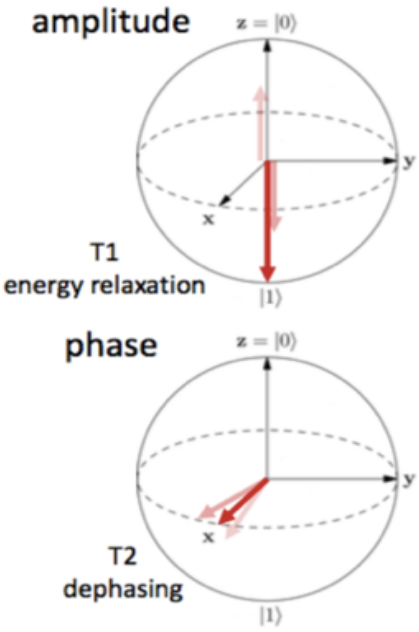

Figure 2: Bloch sphere of energy relaxation and

2.3 Notation

In order to introduce the possible operations on quantum states, it is essential to review the Dirac notation for quantum states. A quantum state $|\Psi\rangle$ is a column vector called **Ket,** while $\langle\Psi|$ is the conjugate transpose (†) row vector known as **Bra**. An arbitrary state in superposition has the following expression:

$$|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{bmatrix}\alpha\\0\end{bmatrix} + \begin{bmatrix}0\\\beta\end{bmatrix} = \begin{bmatrix}\alpha\\\beta\end{bmatrix} \quad \textbf{(2-1)}$$

$$\langle\Psi| = (|\Psi\rangle)^{\dagger} = \begin{bmatrix}\alpha\\\beta\end{bmatrix}^{\dagger} = [\,\alpha^{\dagger}\ \beta^{\dagger}] \quad \textbf{(2-2)}$$

Given two quantum states $|x_1\rangle$ and $|x_2\rangle$ which are defined below, their inner product $\langle x_1| \cdot |x_2\rangle$, sometimes written in short form $\langle x_1|x_2\rangle$, is defined as:

$$|x_1\rangle = \alpha_1|0\rangle + \beta_1|1\rangle \text{ and } \quad |x_2\rangle = \alpha_2|0\rangle + \beta_2|1\rangle$$

$$\langle x_1|x_2\rangle = \left[\,\alpha_1{}^{\dagger}\ \beta_1{}^{\dagger}\right] \cdot \begin{bmatrix}\alpha_2\\\beta_2\end{bmatrix} = \alpha_1{}^{\dagger}\alpha_2 + \beta_1{}^{\dagger}\beta_2 \quad \textbf{(2-3)}$$

The inner product of a state with itself is equal to sum of the probabilities which is always 1.

$$\langle x_1|x_1\rangle = \alpha_1{}^{\dagger}\alpha_1 + \beta_1{}^{\dagger}\beta_1 = |\alpha_1|^2 + |\beta_1|^2 = 1 \quad \textbf{(2-4)}$$

2.4 Quantum Operations

Operations performed on a classical computer rely on logical gates which modify a string of bits. Similarly, in quantum computers the operations are performed using quantum gates to modify the state of the qubits. The laws of quantum mechanics impose some interesting restrictions which will greatly narrow down the types of gates that can be constructed. The first major restriction is that the implementation of quantum system requires the construction of "closed physical system" according to laws of thermodynamics. The second law of thermodynamics, states that in a closed system, operations should not dissipate energy and the total entropy should never decrease. As a result, operations in a quantum system are reversible denoting that no information is lost by the gates [9]. For example, in the classical domain, an AND gate is not reversible since the two input bits cannot be recovered from the single output bit. However, a NOT gate is reversible as the input can be reconstructed from the output. Therefore, a NOT gate can be applied in the quantum domain whereas the construction of quantum AND gate needs some modifications, which will be discussed later on. Another requirement for quantum gates is the need to preserve probability amplitudes. The sum of probabilities should still be equal to 1 after the application of the gate. Given these requirements the operations must be unitary as shown in the following proof.

$$\langle\Psi_1|\Psi_1\rangle = 1 \ \rightarrow \ |\Psi_2\rangle = U|\Psi_1\rangle \text{ where U is the gate applied to the state } \Psi_1$$

$$\langle\Psi_2|\Psi_2\rangle = 1 \ \rightarrow \ (U|\Psi_1\rangle)^{\dagger}U|\Psi_1\rangle = 1 \ \rightarrow \ |\Psi_1\rangle^{\dagger}U^{\dagger}\,U|\Psi_1\rangle = \langle\Psi_1|U^{\dagger}\,U|\Psi_1\rangle = 1$$

$$\rightarrow U^{\dagger} U = I$$

Therefore, quantum gates are unitary matrices with an inverse equal to their conjugate transpose. Now to construct the quantum NOT gate for single qubit, the mapping of the gate can be utilized to find a 2-by-2 unitary matrix performing this operation.

$$\left.\begin{array}{l} \begin{bmatrix} u_1 & u_2 \\ u_3 & u_4 \end{bmatrix} |0\rangle = |1\rangle \rightarrow \quad \begin{bmatrix} u_1 & u_2 \\ u_3 & u_4 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ \begin{bmatrix} u_1 & u_2 \\ u_3 & u_4 \end{bmatrix} |1\rangle = |0\rangle \rightarrow \quad \begin{bmatrix} u_1 & u_2 \\ u_3 & u_4 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{array}\right\} \rightarrow \begin{bmatrix} u_1 & u_2 \\ u_3 & u_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = X \text{ Pauli Gate}$$

The X gate belongs to a group of Pauli operators, which perform rotations of π radians, with two other gates $Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$. Each of the gates perform the rotation around different axis which can be intuitively visualized by the Bloch sphere presented in section 2.7.

2.5 Superposition

To create a superposition of states a special gate H, known as Hadamard gate is applied. The gate transforms the state $|0\rangle$ or $|1\rangle$ into a mixture of the two, where the qubit spends half its time in state $|0\rangle$ and the other half in state $|1\rangle$.

$$H|0\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle = |+\rangle \quad \textbf{(2-5)}$$

$$H|1\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle = |-\rangle \quad \textbf{(2-6)}$$

The states $|+\rangle$ and $|-\rangle$ form another set of basis vectors called the diagonal set. As discussed previously, if the state $|+\rangle$ was to be measured, the result would have been $|1\rangle$ with probability $\left|\frac{1}{\sqrt{2}}\right|^2 = \frac{1}{2}$ or the value $|0\rangle$ with probability $\left|\frac{1}{\sqrt{2}}\right|^2 = \frac{1}{2}$. A common analogy that is often presented for this state is a coin toss, where the outcome of heads or tails is equally probable. It is important to distinguish the difference between the coin in the analogy and the qubit. The qubit is in both states $|0\rangle$ and $|1\rangle$ before the measurement. While the coin has only one side up at any time, it is the tossing action that presents a probabilistic outcome. Another property of a Hadamard gate is

that $H^2 = I$, meaning that if a Hadamard gate operation is applied twice, the qubit retains its original state. Consider the case where H is applied twice to the state $|0\rangle$.

$$H|0\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle = |+\rangle \quad \textbf{(2-7)}$$

$$H|+\rangle = \frac{1}{\sqrt{2}}|+\rangle + \frac{1}{\sqrt{2}}|-\rangle = \frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right) + \frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle\right) \quad \textbf{(2-8)}$$

$$= \left(\frac{1}{2}|0\rangle + \frac{1}{2}|0\rangle\right) + \left(\frac{1}{2}|1\rangle - \frac{1}{2}|1\rangle\right) = |0\rangle$$

This is a very powerful property of quantum systems. Referring back to the coin analogy, this corresponds to the event of un-flipping the tossed coin.

2.6 Bloch Sphere

There exists a graphical representation of qubits using a Bloch sphere, which helps to visualize quantum states in spherical coordinates. Given the constraint that $|\alpha|^2 + |\beta|^2 = 1$, the arbitrary quantum state $|\Psi\rangle$ can be written using a different parametrization.

$$|\Psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle \; where \; 0 \le \theta \le \pi \; and \; 0 \le \phi \le 2\pi$$

The states are represented by a point on a unit sphere in $\mathbb{R}^3$ as shown in the image. The standard basis $|0\rangle$ is in the positive z direction while $|1\rangle$ is in the negative z direction. The positive x axis represents $|+\rangle$, an equal superposition of $|0\rangle$ and $|1\rangle$. The state $|-\rangle$ is on the negative x axis. The Pauli operation gates can be visualized as 180° rotations around the axis of the gate. As an example, the application of Z Pauli gate to $|+\rangle$ will produce $|-\rangle$ state, which is 180° rotation around z axis.

z-axis

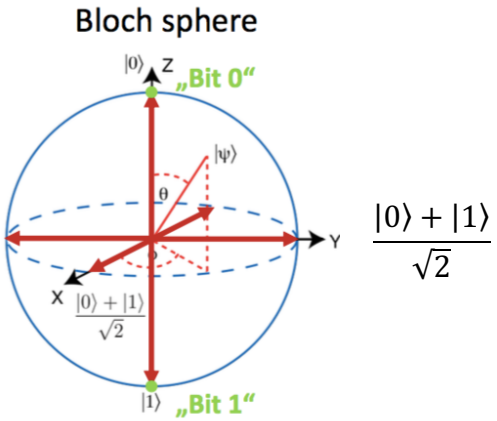

Figure 3: Bloch Sphere [12]

2.7 Multiple Qubit Systems

For a system of n qubits, the dimension of the complex vector space will be 2^n, with the standard basis of binary bit strings $k \in \{0, 2^n - 1\}$. For example, a system with 2 qubits will have the standard basis $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$. Similar to the classical NAND gate, which is universal and can be used to form other logical gates such as AND, OR and XOR, a universal quantum gate exists, known as CNOT or Controlled-NOT gate. This conditional gate has two inputs, control and target qubits. The operation of the gate is to flip the target qubit only if the control qubit is $|1\rangle$, otherwise it will do nothing. Note that CNOT gate has unitary matrix representation which is reversible. The following is the mapping of the 2-qubit CNOT gate.

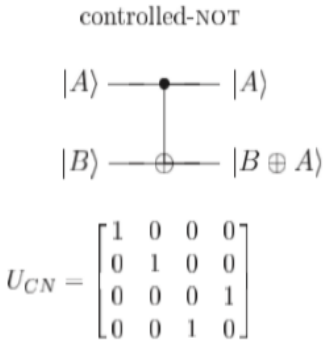

Figure 4: CNOT Gate

$$|00\rangle \rightarrow |00\rangle$$
$$|01\rangle \rightarrow |01\rangle$$
$$|10\rangle \rightarrow |11\rangle$$
$$|11\rangle \rightarrow |10\rangle$$

Moreover, any of the single qubit gates can be transformed to a controlled gate, where one or more qubits are used as control and a gate operation, such as Z Pauli, is performed on the rest of the qubits.

2.8 Entanglement

An interesting quantum circuit that is very simple yet powerful, is the combination of a Hadamard gate in series with a CNOT gate illustrated in figure 5. Initially the qubits are $|00\rangle$. After applying H the qubits become $\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|10\rangle$, and finally the CNOT forms the state $\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$. This famous state, known as EPR pair, showcases the entanglement property of qubits. Measuring the qubits will return $|00\rangle$ with probability of 50% or $|11\rangle$ with probability of 50%. This is a fascinating property, since according to the principles of quantum mechanics, before any measurement is performed the qubits are in both states $|00\rangle$ and $|11\rangle$ at the same time. Now even if the two qubits are physically separated and moved to two different locations, the qubits will remain in the mixture of the two states. Once one of the qubits is measured, both qubits

characteristically collapse into one of the two states ($|00\rangle$ or $|11\rangle$) and given the value of one of qubits the value of the other is revealed.

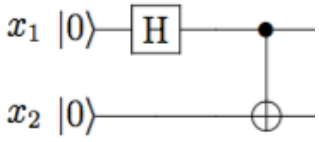

Figure 5: Entanglement Circuit

2.9 Quantum Algorithms

In the study of computation complexity, two general classes of P and NP complexities are defined. The problems that can be solved in polynomial time belong to the P group, while NP problems possess solutions with an exponential run-time at best. Many researchers have attempted to formally answer the open question, that whether the two classes of P and NP are fundamentally different. Due to a lack of a formal proof for classifying complexity of problems, it is possible that many NP problems have polynomial time solutions, which have not been discovered yet. There are many problems under the NP category, which are of great interest in academia and the industry. Feynman who was the first person to propose the idea of quantum computers in 1982, realized that simulation of quantum particles required exponential resources on classical computers [10]. The information required to represent a quantum state grew exponentially, as the quantum system grew larger. He reasoned that employing quantum systems for the simulation of quantum particles, allows one to encode the particle's state information into the system to achieve polynomial runtime. As such, an additional complexity class has been defined known as Bounded-error Quantum Polynomial Time (BQP). Problems in this class require polynomial resources on a Quantum computer with bounded error. Figure 6 shows the set of three complexity class. Note that the hierarchy represented in figure 6 has not been proved. It is only suggested, based on theoretically proven runtimes of certain algorithms such as factoring and simulating quantum mechanical systems.

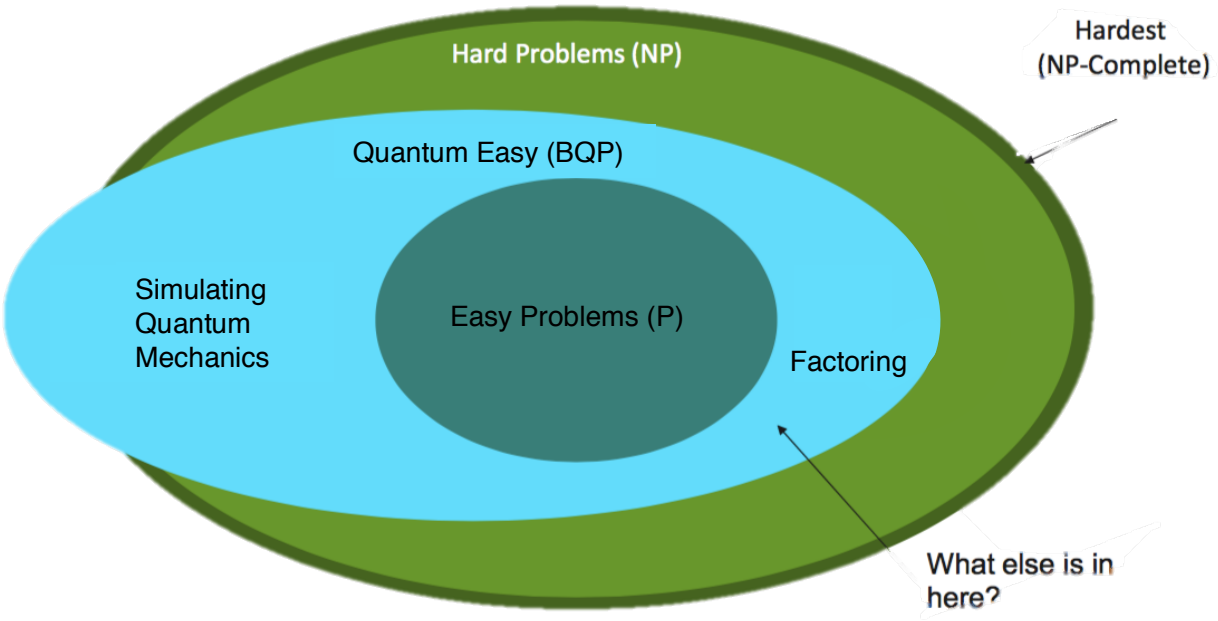

Figure 6: Complexity Classes, P vs. BQP vs. NP [12]

Since then, there have been various applications proposed for quantum computers which range from cryptography to quantum chemistry and machine learning. Currently, the Quantum Algorithm Zoo website has listed 386 papers on quantum algorithms [11]. The challenge in designing these algorithms is to take advantage of the superposition property of the qubits. Assume that an arbitrary algorithm requires an n-bit input and it also produces an n bit output. On a classical computer, computations are performed on the n bit input register with finite intermediate states to obtain the results. Conversely on a quantum computer, the n-bit input, can be stored on n-qubits. Once in superposition the state of the n qubit system requires 2^n complex numbers to express its amplitudes. The 2^n complex amplitudes are sufficient to represents all of the possible states of n-bit classical register at once. This property enables many quantum algorithms to surpass their classical counterparts in terms of computational complexity [12].

3. Quantum Computing System Architecture

The design and production of quantum computing systems is still in its infancy stages. Recently, interest in building such systems has greatly escalated. Due to the cybersecurity threats posed by the potential arrival of error-tolerant quantum technology, national government security agencies have long been the major actors in pushing the frontier of quantum development. However, recently industry leaders have joined the race for building the first commercial general-purpose quantum computer. Tech companies along with startups and university research groups have invested heavily in building these devices. As such, there has been an exponential rise in the number of engineering papers published in the area of quantum computing, presented in figure 7.

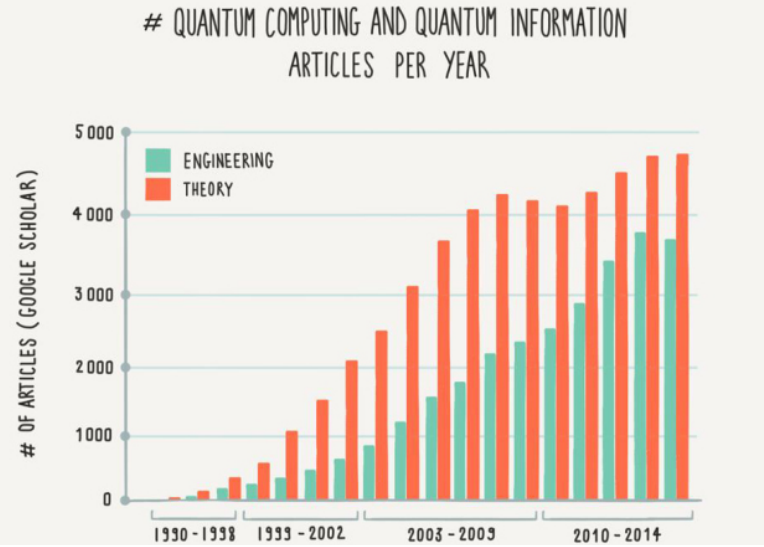

Figure 7: Increase in quantum research papers over the last two decades

3.1 Physical Quantum Systems

The qubit can be implemented in many different physical systems such as an electron orbiting a single atom, polarization of a photon or alignment of nuclear spin in a uniform magnetic field [13]. The successful realization of the qubit has been achieved in many systems; the mature implementations include superconducting qubits, trapped-ion, solid-state spin, nuclear spin, non-linear photonic, and neutral atom qubits.

Currently the two most popular architectures under research and development are ion trapped qubits and superconducting transmon. Trapped ion quantum computers, realize qubits as a physical system of ions (charge particles) suspended in vacuum within an electromagnetic field. Laser technology is used to control the ions and perform operation with qubits. This allows for all the

qubits to communicate with each other. The other physical system used by IBM's prototype machines relies on superconducting transmon qubits, composed of superconducting elements such as niobium and aluminum, fabricated on a silicon wafer. This approach is widely adopted in industry, where in addition to IBM, Google, Intel, Rigetti and Quantum Circuits have also employed this same technique [14]. This architecture leverages the many advances and innovations achieved in silicon fabrication over the digital age. It is required for a superconducting qubit to be cooled down to extremely low temperatures close to 15 milliKelvin in a dilution refrigerator. These conditions ensure control over the state of the qubit, by eliminating ambient noise and heat, which can excite the qubit. Superconducting qubits require fixed circuitry elements for implementing 2 qubits gates. Therefore, they are only allowed to communicate with their immediate neighbors, which is a major drawback. Ion trapped qubits have better connectivity and also coherence, by conserving the state of qubits for relatively long periods of time. Operations performed with superconducting transmon qubits are much faster than ion trapped qubits. Furthermore, the scalability, in terms of the number of qubits, for a transmon architecture appears to be much easier than ion trapped systems [15]. Table 1 summarizes the major difference between these two architectures.

Architecture	**Ion Traps** [15]	**Superconducting Qubits** [15]
Qubits realized as :	Spin of ionized atoms trapped in a vacuum of electromagnetic field	Current that flows through superconducting elements fabricated on silicon wafer
Max number of qubits	5	72
Lifetime of qubit (T2)	50 seconds	50 micro-seconds
Best gate precision (2-gate fidelity)	99.9%	99.4%
Advantages	•Better connectivity •Long-lived qubit memory and coherence	•Fast gate time •Exploits advances in silicon fabrication (scalable)
Disadvantages	•High power to perform gates	•Needs to be kept in extreme low temperatures (~10mk) •Poor qubit T2 lifetime

Table 1 Difference between Ion Trap and Superconducting Qubit machines

The remainder of this report will focus on superconducting qubits due to two main reasons. The first being the superior scalability of superconducting qubits has made them the prime contender for future commercial quantum computers. Additionally, all of the IBM quantum processors have been built on this architecture. These processors have been employed for a number of experiments in this report through the IBM cloud quantum computing services [16].

3.2 Superconducting Qubits

Figure 8 showcases the schematic and placement of qubits on an example 2-by-2 lattice quantum processor. The main components of the device are superconducting transmon qubits (Q_1-Q_4), readout resonators (R_1-R_4) and coupling buses (B_{12}, B_{23}, B_{34}, B_{41}). A blowup picture of the transmon qubit is also shown, depicting the capacitor geometry containing the Josephson junction. Superconducting readout lines are used to transmit microwave signals at frequency of ω_{Mi} to read out the state of the qubit. Additionally, they are used to send single- and two-qubit control gate signal at frequency ω_i. The coupling buses provide connectivity between neighboring qubits.

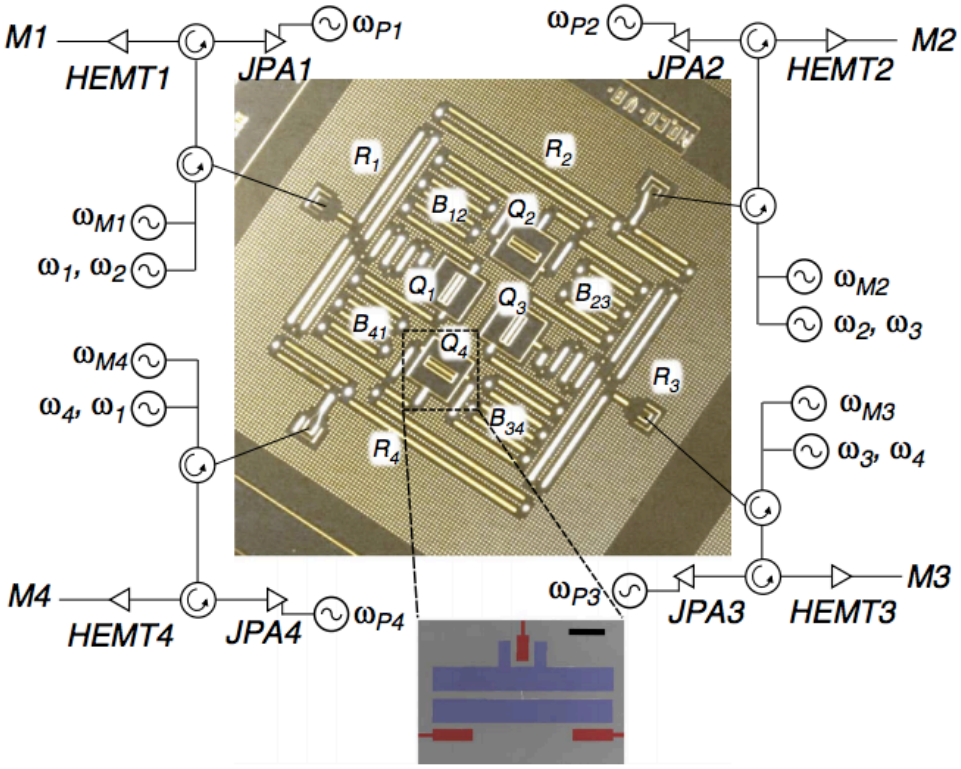

Figure 8: Quantum Processor Layout

3.3 Two Qubit Gates

In the current architecture of IBM quantum processors, the qubits are laid on a planer lattice structure. The shape of the structure depends on the number of qubits on the processor. The significance of the lattice shape it that the two qubit gates are only realized through coupling buses shown in figure 8 between neighboring qubits. Therefore, the lattice structure greatly affects the connectivity of the qubits and the performance of two-qubit gates.

The implementation of the CNOT gates in the hardware is directional, where between any two neighbors, one always acts as the control qubit and the other as the target. The CNOT gate is grounded on a cross-resonance interaction between qubits, where the interaction is stronger if the high frequency qubit is the control and the lower frequency one is the target. Figure 9 demonstrates the connectivity graph for 16 qubit IBM QX5 processor, where the arrows show the direction of CNOT gates between neighboring qubits.

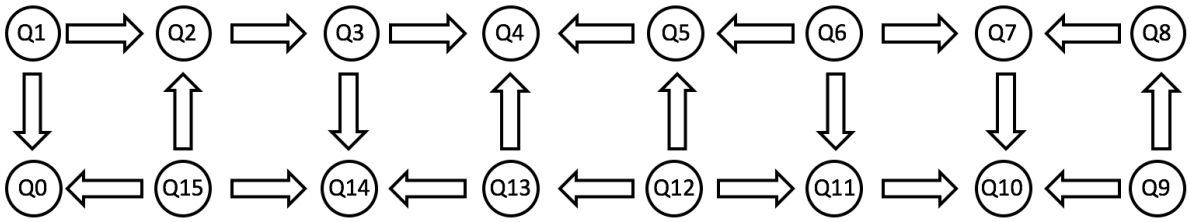

Figure 9: Presents the connectivity graph of qubits on IBM QX5 quantum processor.

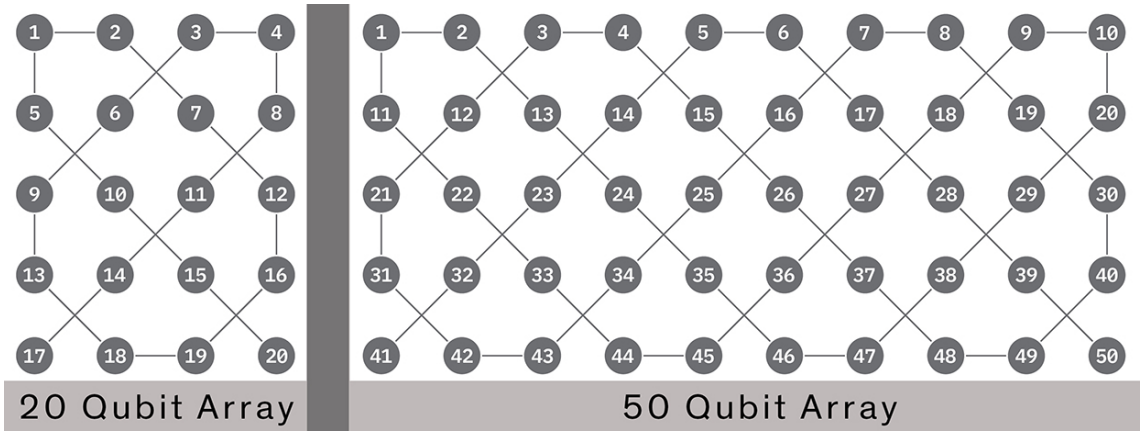

Figure 10: Connectivity graph of 20-qubit and 50-qubit quantum processors under development.

Quantum algorithms assume full connectivity between qubits where many qubits need to interact with one another. To overcome the challenge of the limited connectivity and directional gates, the

IBM QISKit programming package, performs a greedy search to find an optimized sequence of SWAP gates. In order the perform CNOT operations between two non-neighboring qubits, the sequence of SWAP gates copy the state of the qubit to two neighboring physical qubits with the correct direction of CNOT gate between them. The SWAP operations greatly increase the complexity of any algorithm.

4. Grover's Algorithm

Grover's algorithm tackles the problem of unstructured search, where a database of N elements is provided, and the objective is to find a target element in the database. The major strength of this quantum algorithm is its ability to take in a generic list, where no restrictions on the format or ordering of the list have been imposed. A formal definition of the problem is stated below.

4.1 Problem statement

Unstructured Search: Given a set $X = \{x_1, x_2, x_3, \dots, x_N\}$ with N entries and an oracle function $f: X \rightarrow \{0,1\}$, the goal is to find a target element x^* in X which satisfies $f(x^*) = 1$.

4.2 Oracle function

The oracle function is required to identify the target element in a query. In the classical setting, the worst-case runtime of this problem is O(N) where the target is the last element in the list, and all elements have to be checked with the oracle function. Now in the quantum domain a quadratic speedup is achieved by reducing the worst-case runtime to $\sqrt{N}$ queries [17]. To study and analyze the performance of Grover's algorithm, the unstructured search problem is simplified by three assumptions.

1- The target element x^* is unique.

2- The number of elements in the database is power of 2, $N = 2^n$

3- The elements in the database are indexed as n-bit strings $\{0,1\}^n$.

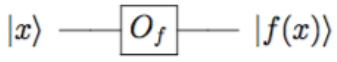

Figure 11: Oracle Gate [9]

Note that the assumptions listed above are without loss of generality to simplify the forthcoming analysis. The idea is to prepare a system of qubits which will represent the indices of the elements in the list. Similar to the classical approach an oracle gate will be applied to the qubits to evaluate whether each entry is the target or not. A diagram of an oracle gate mapping is presented in figure 11. This oracle mapping is clearly not reversible or unitary, as the input and output dimensions do not match. Thus, to construct the oracle gate a clever trick is utilized and it is defined as follows:

$$O_f^{\pm}|\mathrm{x}\rangle = (-1)^{f(x)}|\mathrm{x}\rangle = \begin{cases} |\mathrm{x}\rangle & if\ f(x) = 0 \\ -|\mathrm{x}\rangle & if\ f(x) = 1 \end{cases} \quad \textbf{(4-1)}$$

The oracle reverts the sign of the target state, while all the other states remain intact. Given the oracle gate it is possible to study the overall quantum circuit for Grover's algorithm presented in figure 12.

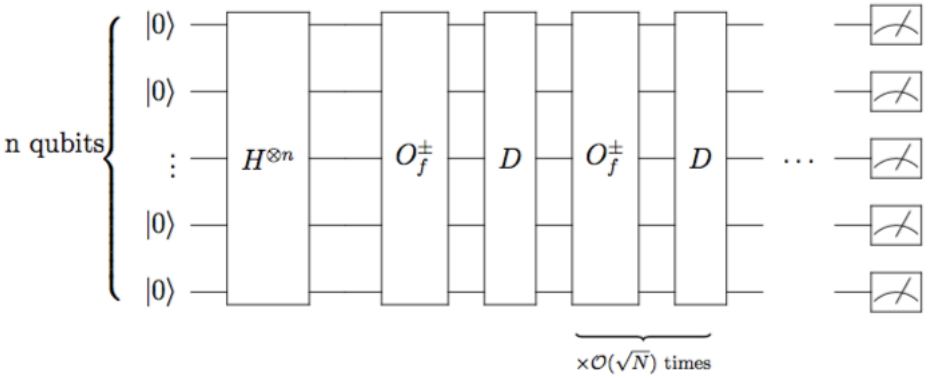

Figure 12: Overview of Grover's Circuit [9]

First the qubits are all initialized in $|0\rangle$ basis state. Then the Hadamard gate is applied to all qubits, to produce uniform superposition of 2^n elements. This state is equivalent to a uniform probability for each of the items in the list to be the target element x^*.

$$\text{Amplitude of each state} = \frac{1}{\sqrt{N}} \qquad \text{Probability of each state} == \frac{1}{N}$$

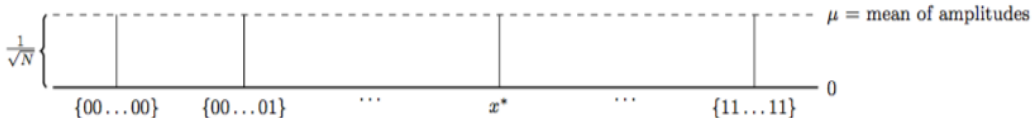

Figure 13: Initial Superposition of States [9]

Now if the oracle gate is applied to this uniform superposition, the amplitude of the target x^* is negated, while the rest of the states will have the same amplitude.

$$Ampliude = \frac{1}{\sqrt{N}}|x\rangle \text{ for } each\ state\ that\ is\ not\ the\ target\ x^*$$

$$Amplitude = -\frac{1}{\sqrt{N}}|x\rangle \quad \text{for the target state } x^*$$

Figure 14: Superposition of States after the Oracle is applied [9]

4.3 Grover's Diffusion Gate

The motivation for next step is to maximize the amplitude of the target state x^* relative to all the other states. To accomplish this, a new operation called the Grover Diffusion gate is applied. The

gate flips the state amplitudes about the mean of all the amplitudes in superposition. The mean is defined as $\mu = \frac{1}{N}\sum_x a_x \; for \; all \; x \in \{0,1\}^n$ where a_x is the amplitude of each state. Currently the mean is equal to $\mu = \frac{1}{N}\sum_x a_x = \frac{1}{N}\left(\frac{N-1}{\sqrt{N}} - \frac{1}{\sqrt{N}}\right) = \frac{1}{N}\left(\frac{N-2}{\sqrt{N}}\right) \approx \frac{1}{\sqrt{N}}$. The negative amplitude of the target has slightly reduced the mean from $\frac{1}{\sqrt{N}}$, which with the assumption that N is large can be ignored. Next by applying the diffusion gate, the amplitudes are transformed according to the following mapping.

$$\sum_x a_x |x\rangle \longmapsto \sum_x (2\mu - a_x)|x\rangle \qquad \textbf{(4-2)}$$

This operation transforms the positive and negative amplitudes in the superposition as follows:

$\frac{1}{\sqrt{N}}|x\rangle \longmapsto \left(\frac{2}{\sqrt{N}} - \frac{1}{\sqrt{N}}\right)|x\rangle = \frac{1}{\sqrt{N}}|x\rangle$ where $x \neq x^*$ remain approximately the same

$-\frac{1}{\sqrt{N}}|x\rangle \longmapsto \left(\frac{2}{\sqrt{N}} + \frac{1}{\sqrt{N}}\right)|x\rangle = \frac{3}{\sqrt{N}}|x\rangle$ where $x = x^*$ is tripled compared to the initial value

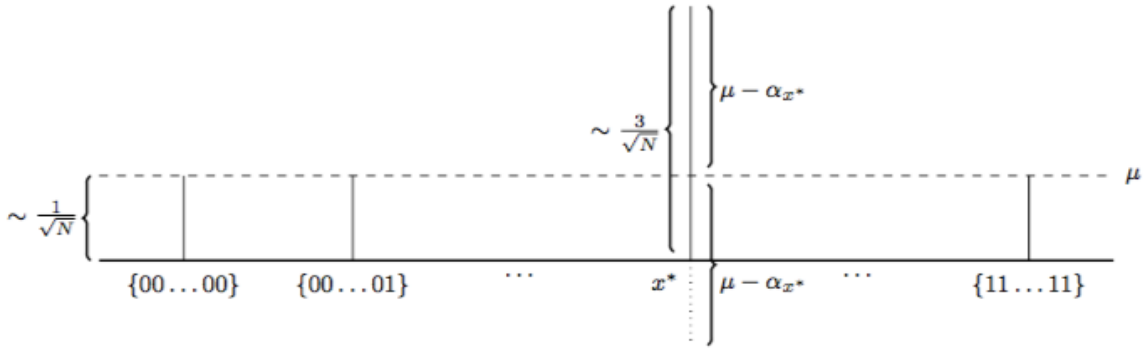

Figure 13: Superposition Of States After First Iteration [9]

Effectively the gate maximizes the amplitude of the target, while the rest of the states see a very slight decrease in their amplitude. Note that if the qubits were measured at this stage, the probability of the superposition collapsing to the target is higher than each of the other states ($\frac{3}{\sqrt{N}} > \frac{1}{\sqrt{N}}$). However, depending on the value of N, the probability is not high enough to guarantee that the target is chosen. Therefore, the amplitude of the target must be maximized further by repetitively applying the oracle and Grover diffusion gate. According to figure 16, after the second iteration the amplitude increases to approximately $\frac{5}{\sqrt{N}}$. The amplitude of the target a_{x^*} keeps increasing with each iteration of the two operators, oracle and diffusion gate. However, this

increase cannot be indefinite, as intuitively the probability of the target state a_{x^*} should not become bigger than 1. In fact, the amplitude starts to go down when a_{x^*} reaches a certain large value. Since the oracle operator will flip the sign of a_{x^*} to produce a negative large value which will ultimately lower the mean of all the amplitudes to a negative value. At this point, the Grover diffusion gate will use the negative mean to reduce the amplitude of the target a_{x^*}. This can be easily demonstrated by using the gate mappings to calculate the success probability after each iteration.

Figure 14: Superposition State After Second Iteration [9]

Let $a_{x^*}^t$ be the amplitude of target state and let a_x^t be the amplitude of all the other states at iteration t. Each iteration is defined as an execution of the oracle and Grover's diffusion gate. Initially the amplitudes are equal to

$$a_{x^*}^0 = a_x^0 = \frac{1}{\sqrt{N}} \quad \textbf{(4-3)}$$

$$\mu^t = \frac{-a_{x^*}^t + a_x^0(N-1)}{N} \quad \textbf{(4-4)}$$

$$a_{x^*}^{t+1} = 2\mu^t + a_{x^*}^t \quad \textbf{(4-5)}$$

$$a_x^{t+1} = 2\mu^t - a_x^t \quad \textbf{(4-6)}$$

Using the above formulas, the success rate at each iteration of the algorithm is calculated for a database of length 4, 16, 64 and 256. Examining the figure 16, the oscillatory effect on Grover's success rate is evident as the number of iterations grows.

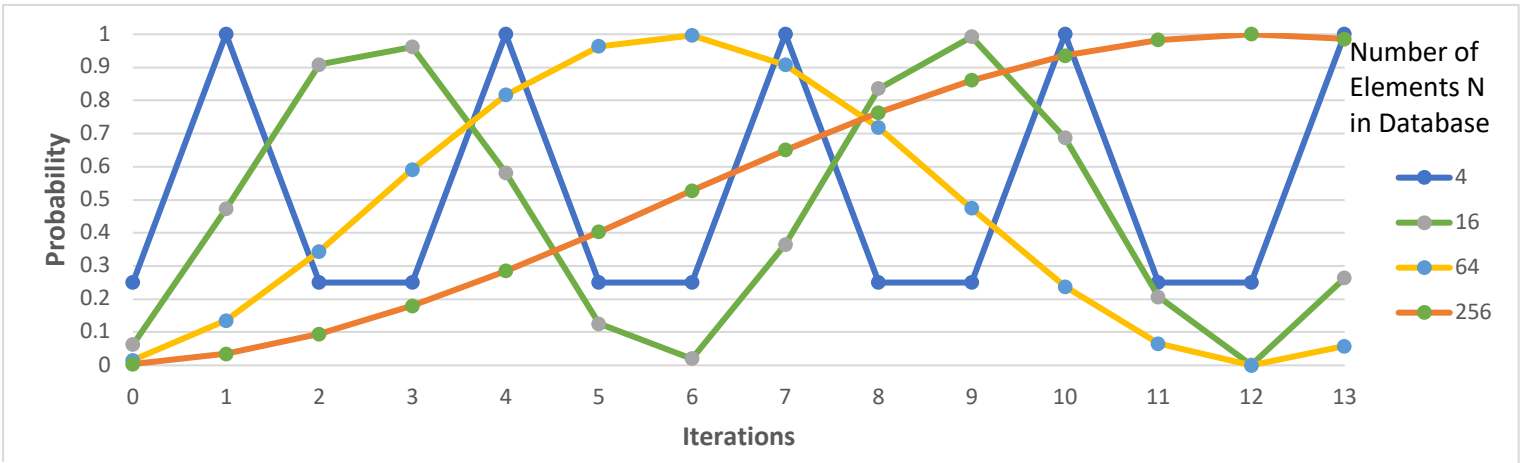

Figure 15: Grover Success Rate vs Number of Iterations

It can be shown that the number of iterations required for the probability to be close to 1 is approximately $k = \frac{\pi}{4}\sqrt{N}$ [18]. Note that the number of iterations must be an integer and therefore k is rounded down to attain the lowest number of iterations after which the probability starts to decline. The probability of finding x* where $f(x^*) = 1$ is estimated to be $P = \sin^2((2k+1)\sin^{-1}\left(\frac{1}{\sqrt{N}}\right))$ [18]. Table 2 presents the success probability based on the predicted number of iterations required for some sample database lengths. Although the success rate of finding the target element is fairly high, it is not exactly equal to 1 and the algorithm can possibly return a false state. Nevertheless, note this error can be conveniently resolved by validating the measured state with the oracle function. In the case that the target element is not returned, the algorithm can be repeated. It is expected to find the target by repeating the algorithm $\frac{1}{P}$ times on average.

Success Probability	Number of Iterations Required	k	N
0.5	1	1.11	2
1	1	1.57	4
0.946	2	2.22	8
0.961	3	3.14	16
0.996	6	6.28	64
0.996	8	8.88	128
0.986	13	12.56	256
0.995	18	17.77	512

0.999	25	25.13	1024

Table 2: Success probability of Grover's algorithm for sample database lengths

4.4 Grover's Circuit

The next step is to go over the procedure for constructing the Grover diffusion gate. The unitary operation of the gate is expressed as $U_d = 2|+^{\mathrm{n}}\rangle\langle+^{\mathrm{n}}| - I$. Recall the definition of the diagonal basis vectors as $|+^{\mathrm{n}}\rangle = \frac{1}{\sqrt{N}}\sum_{x=0}^{N-1}|\mathrm{x}\rangle$, the expression for U_d can be expanded out to its unitary matrix form.

$$U_d = 2\,|+^{\mathrm{n}}\rangle\langle+^{\mathrm{n}}| - I = 2\begin{bmatrix}\frac{1}{\sqrt{N}}\\ \frac{1}{\sqrt{N}}\\ \cdots\\ \frac{1}{\sqrt{N}}\end{bmatrix}\begin{bmatrix}\frac{1}{\sqrt{N}} & \frac{1}{\sqrt{N}} & \ldots & \frac{1}{\sqrt{N}}\end{bmatrix} - I \qquad \textbf{(4-7)}$$

$$= 2\begin{bmatrix}\frac{1}{\sqrt{N}} & \frac{1}{\sqrt{N}} & \cdots & \frac{1}{\sqrt{N}}\\ \frac{1}{\sqrt{N}} & & & \frac{1}{\sqrt{N}}\\ \vdots & & \ddots & \frac{1}{\sqrt{N}}\\ \frac{1}{\sqrt{N}} & & & \frac{1}{\sqrt{N}}\end{bmatrix} - I = \begin{bmatrix}\frac{2}{\sqrt{N}} - 1 & \frac{2}{\sqrt{N}} & \cdots & \frac{2}{\sqrt{N}}\\ \frac{2}{\sqrt{N}} & & & \frac{2}{\sqrt{N}}\\ \vdots & & \ddots & \frac{2}{\sqrt{N}}\\ \frac{2}{\sqrt{N}} & & & \frac{2}{\sqrt{N}} - 1\end{bmatrix}$$

Using the following three identities, covered in the notation section of this report, the expression for U_d can be simplified further.

1) $|+^{\mathrm{n}}\rangle = \mathrm{H}^{\mathrm{n}}|0^{\mathrm{n}}\rangle$, 2) $\langle+^{\mathrm{n}}| = (\,\mathrm{H}^{\mathrm{n}}|0^{\mathrm{n}}\rangle\,)^{\dagger}$, 3) $H^nH^n = I$

$$U_d = 2|+^{\mathrm{n}}\rangle\langle+^{\mathrm{n}}| - I = 2(\,\mathrm{H}^{\mathrm{n}}|0^{\mathrm{n}}\rangle(\,\mathrm{H}^{\mathrm{n}}|0^{\mathrm{n}}\rangle\,)^{\dagger}) - H^nH^n = H^n\,(2\,|0^{\mathrm{n}}\rangle\langle0^{\mathrm{n}}| - I)\;H^n \qquad \textbf{(4-8)}$$
$$= -H^n\;\;(I - 2\,|0^{\mathrm{n}}\rangle\langle0^{\mathrm{n}}|)\;H^n\;\;\rightarrow\;H^n\,(I - 2\,|0^{\mathrm{n}}\rangle\langle0^{\mathrm{n}}|)\;H^n$$

Note that in the last step, the negative sign in the equation is neglected, since the global phase of the qubits does not matter, and it is only the relative phase that is of significance. The expression in the brackets, can be expanded to make a unitary matrix as follows:

$$(I - 2\,|0^{\mathrm{n}}\rangle\langle0^{\mathrm{n}}|)\;= I_{nxn} - \begin{bmatrix}2 & 0 & \cdots & 0\\ 0 & 0 & & \\ \vdots & & \ddots & \vdots\\ 0 & & \cdots & 0\end{bmatrix} = \begin{bmatrix}-1 & 0 & \cdots & 0\\ 0 & 1 & & \\ \vdots & & \ddots & \vdots\\ 0 & & \cdots & 1\end{bmatrix} \qquad \textbf{(4-9)}$$

This unitary matrix can be realized through specific series of gates $X^nZ_CX^n$ where X^n is the Pauli NOT gate applied to all the qubits and Z_C is the controlled Z operator [19].

$$X^n Z_C X^n = \begin{bmatrix} 0 & \dots & 0 & 1 \\ \vdots & & 1 & 0 \\ & \iddots & & \vdots \\ 1 & & \cdots & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & & \\ \vdots & & \ddots & \vdots \\ 0 & & \cdots & -1 \end{bmatrix} \begin{bmatrix} 0 & \dots & 0 & 1 \\ \vdots & & 1 & 0 \\ & \iddots & & \vdots \\ 1 & & \cdots & 0 \end{bmatrix} = \begin{bmatrix} -1 & 0 & \cdots & 0 \\ 0 & 1 & & \\ \vdots & & \ddots & \vdots \\ 0 & & \cdots & 1 \end{bmatrix} \quad \textbf{(4-10)}$$

$$H^n \left(I - 2 \left| 0^n \right\rangle \left\langle 0^n \right| \right) H^n = H^n \left(X^n Z_C X^n \right) H^n \quad \textbf{(4-11)}$$

Finally putting all the pieces together, the complete circuit for Grover's algorithm can be formed as displayed in the following diagram.

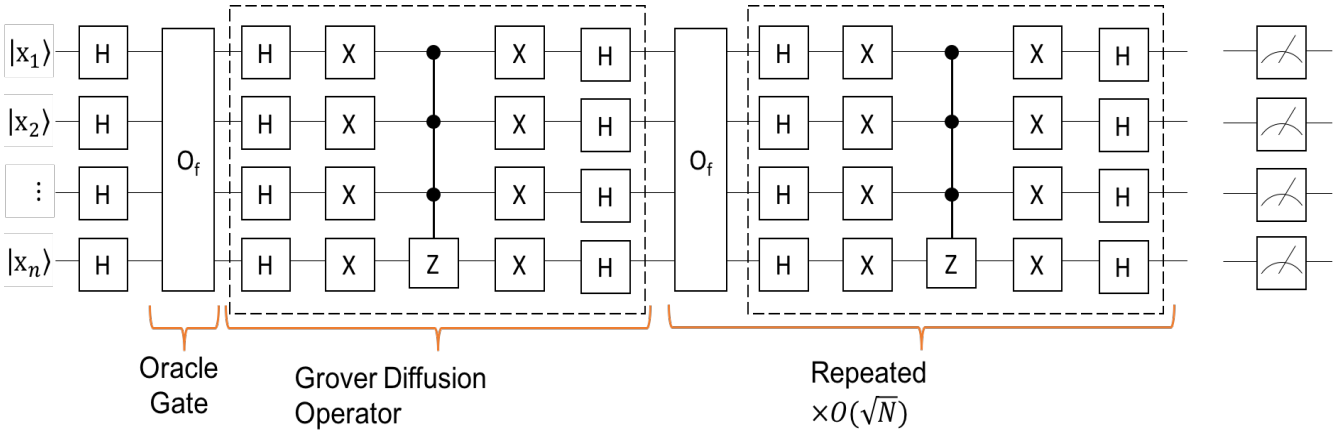

Figure 16: Complete circuit for Grover's algorithm

4.5 Grover's Circuit Implementation on IBM Q

The circuit in figure 18 was adapted and implemented on the IBM Quantum experience. Figure 19 shows the full implementation, where the numbered boxes correspond to the following list of steps:

1. Induce Superposition with Hadamard Gate
2. Oracle Gate
3. Grover's Iteration Gate
4. Measurement

Steps 2 and 3 have been repeated to increase the probability of the measurement resulting in the target step. This implementation uses 7 qubits, where 3 of which are ancillary qubits. The oracle function used, searches for the state 1110. Consider step 2 shown in the blow-up image of figure 19, the two X gates search for 0 in the first qubit of the target state. The CCX (a, b, c) gates in the V structure correspond to AND gate between all qubits.

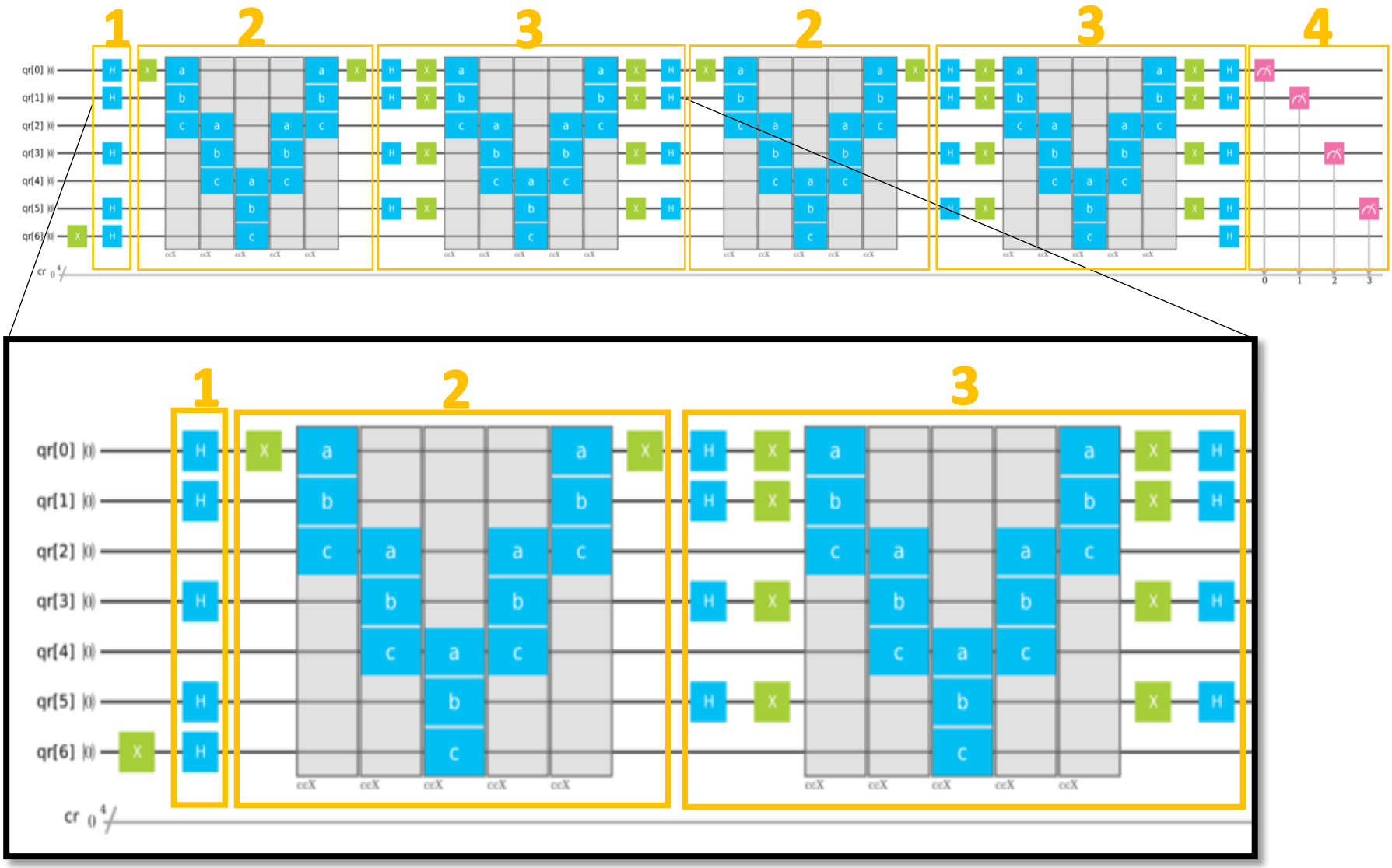

Figure 17 Implementation of Grover's circuit on IBM Quantum Experience

4.6 IBM Q Grover Results

The results of the experiment are presented below. Figure 20 shows the results of the 1024 measurement after the first iteration of steps 2 and 3. As expected, the target state 1110 has the highest probability of 48.7%, while all the other states are fairly equal with a probability less than 5%.

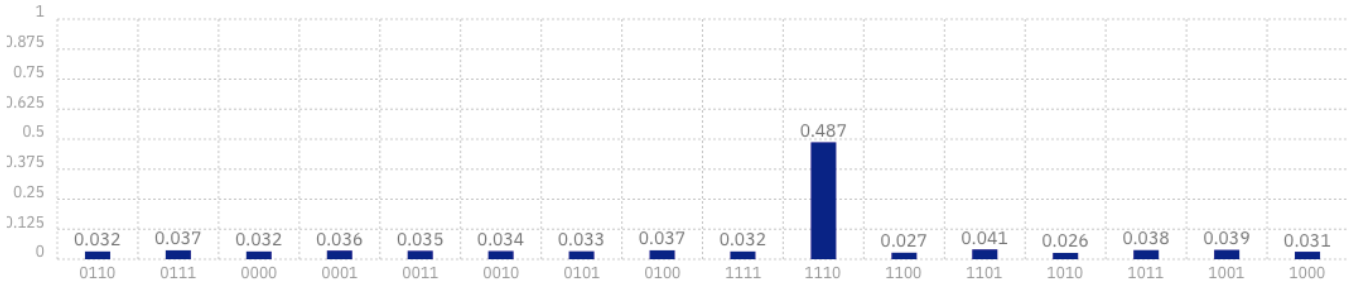

Figure 18 Grover circuit results with single iteration

The plot in figure 21 shows the 1024 measurement results after the second iteration of the oracle gate and Grover's diffusion gate. Comparing the two, it is evident that the second iteration, almost doubles the probability of the target state, while reducing the probability of all the other states.

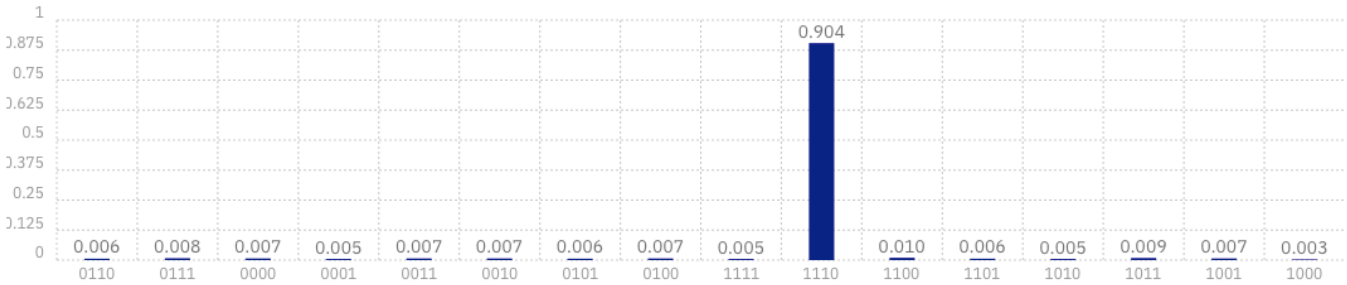

Figure 21 Grover circuit results with two iterations

The constructed circuit, is only a demonstration of Grover's algorithm [20] where the target state is unknown before constructing the circuit in a small search space. Grover's search becomes only useful given a black box oracle function where the target is unknown, and the search space is fairly large. However, since it only provides a quadratic speedup, it does not provide a significant improvement for the class of problems that require exponential resources.

5. Shor's Algorithm

The focus of Shor's algorithm is factoring large numbers. A task which has been found to be very challenging in terms of time and computation resources for classical computers. Conversely multiplication of two large numbers is accomplished effortlessly on classical computer. In fact, many of the cryptography algorithms exploit the strength of computers in multiplying large numbers and their weakness to factor them back i.e. one way functions. A widely used public key cryptography protocol known as the RSA, selects two large prime numbers p, q and multiplies them to generate a public key $N = p \times q$. The number N is available to the public with the assurance that recovering p and q requires exponential time or computational resources with respect to the number of digits in N. As an example, RSA-768 is the largest RSA key factored in the year 2009 with 232 decimal digits and (768 bits). The time and resources spent on this task was "2000 years of computing on a single-core 2.2 GHz AMD Opteron-based computer" equivalent to the order of 2^{67} instructions [21].

5.1 Complexity of Factoring

Given a large number N with d digits, the simplest classical algorithm will check all the primes p starting from 1 to $\sqrt{N}$ to see if N is divisible by any value of p. The worst-case runtime of this brute force algorithm is $O(\sqrt{N}) = O\ (e^{d})$ exponential in the number of digits. The best classical algorithm known to date is the general number field sieve which achieves $O(e^{\sqrt[3]{d}})$ and is still of exponential order. In the quantum domain, a great speedup is achieved using Shor's algorithm, capable of solving the factorization problem in polynomial time with respect to number of digits. Consequently, the availability of powerful general-purpose quantum computers is a great threat to RSA and other public key cryptography protocols.

Shor's algorithm can be broken down to two fundamental procedures, a classical reduction part and period finding quantum algorithm.

5.2 Shor's Motivation

The official problem statement is that given a large positive odd integer N, find the prime factorization pair (p and q) which make up N. It is important to note that the prime factorization

pair is unique for any number. Also, N is chosen as an odd number, since otherwise 2 would be one of the prime factors. Instead, of directly solving the factorization problem, mathematicians discovered a parallel problem in 1970, where the solution would lead to the discovery of the prime factors. The problem is finding the period of modular exponential function which is defined below.

Modular Exponential Function: $F(x) = a^x \ mod \ N, where \ a \ is \ an \ integer < N$ and $N \ is \ randomly \ chosen, a \ and \ N \ must \ be \ coprime \ (a \ mod \ N \neq 0) \ and \ x \ \epsilon \ \mathbb{N}$.

This function returns a periodic sequence over values of x for any value of a and N.

$$F(x + r) = F(x) \rightarrow \ a^{x+r} \ mod \ N = a^x \ mod \ N \qquad \textbf{(4-1)}$$

The periodic property of the function is easily demonstrated through an example such as the sequence of powers of 2 where $a = 2, N = 21$:

$$2^x = 1, 2, \ 4, \ 8, \ 16, \ 32, \ 64, 128, 256, 512, \ldots$$
$$2^x \ mod \ 21 = 1, 2, \ 4, \ 8, \ 16, \ 11, \quad 1, \quad 2, \quad 4, \quad 8, \ldots$$

Evidently the period in the example above is 6. Note that the first element in series is always 1 since $a^0 = 1$. Moreover, since the function is periodic $F(0) = F(0 + r) = a^r \ mod \ N = 1$ $\rightarrow \ (a^r - 1) \ mod \ N = 0$. This means that $(a^r - 1)$ is a multiple of N. The values of p and q can be determined by finding the smallest integer r satisfying $(a^r - 1) \ mod \ N = 0$ which is then factored out by the following identity:

$$a^r - 1 = (a^{r/2} - 1) \ (a^{r/2} + 1) \qquad \textbf{(4-2)}$$

Examining equation (4-2), it is known that $(a^{r/2} - 1)$ is not a multiple of N, since otherwise the period of $F(x)$ would have been $r/2$. Assuming that $(a^r + 1)$ is also not a multiple of N, p and q should each be a prime factor of one of the two terms $(a^{r/2} - 1) \ (a^{r/2} + 1)$. The last step is to use Euclid's algorithm for calculating the greatest common divisor between N and each of the terms: $gcd \ (a^{r/2} + 1, N)$ and $gcd(a^{r/2} - 1, N)$. In the unfortunate event that $(a^r + 1)$ is a multiple of N, the value of a is tossed and another a is chosen.

Continuing the example presented above, where $N = 21, a = 2 \ and \ r = 6$, the prime factors can be evaluated as follows:

$$(a^{r/2} - 1) = (2^3 - 1) = 7 \text{ and } (a^{r/2} + 1) = (2^3 + 1) = 9$$

$$gcd\,(7, 21) = 7 \text{ and } gcd\,(9, 21) = 3$$

$$N = p \times q = 3 \times 7$$

As demonstrated through the example above, it can be concluded that solving the period finding problem leads to a solution for the factorization problem. Shor's algorithm provides a quantum subroutine for finding the period of a modular exponential function which will complete the solution. Notice that this is the only step which requires quantum computation and all the other calculations can be performed on a classical computer.

To summarize the procedure discussed, an outline of the algorithm for factorizing N is listed below.

1. Select a random number a between 1 and $N - 1$
2. Check if $\gcd\,(a,\, N - 1) = 1$ using Euclid's algorithm and continue to step 3 if true, otherwise a is cofactor of N which can be used to find $p\ and\ q$
3. Employ the quantum subroutine to find the period r of $F(x) = a^x\ mod\ N$
4. If r is odd or $(a^r + 1)\ mod\ N = 0$ go to step 1 and pick another a.
5. $p = \gcd\left(a^{r/2} + 1, N\right)$ and $q = gcd(a^{r/2} - 1, N)$

5.3 Period Finding

A fundamental element of the period finding algorithm is the Quantum Fourier Transform (QFT). The QFT is the quantum equivalent to the Discrete Fourier Transform in the classical domain. It is very often used in phase estimation problems. Recall that the DFT maps a sequence of complex numbers $x_n = x_0, x_1, \dots, x_{n-1}$ to another sequence of complex number $y_n = y_0, y_1, \dots, y_{n-1}$ according to the following expression:

$$y_k = \sum_{j=0}^{N-1} x_j \cdot \omega_N^{kj} = \sum_{j=0}^{N-1} x_j \cdot e^{i2\pi kj/N}, k = 0, 1, 2, \dots, N - 1 \qquad \textbf{(4-3)}$$

The QFT applies a linear Discrete Fourier transform on the amplitudes of the quantum state:

$\sum_i x_i\,|\mathrm{i}\rangle \longmapsto \sum_i y_i\,|\mathrm{i}\rangle$ where $y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j \cdot \omega_N^{kj} = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j \cdot e^{i2\pi kj/N}, k = 0, 1, \dots, N - 1$

Note that in the transformation above only the amplitudes were affected. Moreover, the mapping for pure state can be expressed as:

$$|\mathrm{x}\rangle \longmapsto \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} \omega_N^{xy} \, |\mathrm{y}\rangle \tag{4-4}$$

Then, the unitary matrix that implements the QFT can be written as:

$$U_{QFT} = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \omega_N^{xy} \, |\mathrm{y}\rangle \langle \mathrm{x}| \tag{4-5}$$

$$= \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \omega_n^3 & \cdots & \omega_n^{N-1} \\ 1 & \omega_n^2 & \omega_n^4 & \omega_n^6 & \cdots & \omega_n^{2(N-1)} \\ 1 & \omega_n^3 & \omega_n^6 & \omega_n^9 & \cdots & \omega_n^{3(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{N-1} & \omega_n^{2(N-1)} & \omega_n^{3(N-1)} & \cdots & \omega_n^{(N-1)(N-1)} \end{bmatrix} |\mathrm{y}\rangle\langle \mathrm{x}|$$

The square weight matrix in the equation (4-5) is exactly equivalent to the transformation matrix for an N-point DFT. The time complexity for applying the DFT transformation matrix is of $O(N^2)$, since the size of the matrix is N-by-N. Classically the best optimized algorithm for implementing the DFT, is the Fast Fourier Transform (FFT). It computes the transform for a vector N points with time complexity of $O(NlogN)$. Moving to the quantum domain, an exponential improvement in the computation time can be achieved. This is due to the fact that a system of dimension N, can be realized by n qubits, where $2^n = N$. The most naïve implementation of QFT circuit will have complexity of $O(n^2) = O(\log^2 N)$. Nevertheless, there is a great limitation associated with this exponential speedup. Classically the DFT takes as input a vector of N numbers and outputs a transformed vector of N numbers. With the QFT, the N numbers are captured as the amplitudes of n qubit system and as the transformation is applied, a new n qubit system is obtained. The laws of Quantum Mechanics, deny direct access to the N amplitude values captured in the superposition. Recall that measuring a quantum system in superposition, gives access only to a sample state in the superposition.

5.4 Interference

A key feature of the QFT, is its role in interference of superposition amplitudes. Assume that superposition state $\alpha_i|\mathrm{j_i}\rangle$ is a periodic sequence of dimension M, where $i = 0, 1, \ldots, M-1$. A diagram of this system is represented in figure 22. The states that are an integer multiple of r have nonzero amplitudes, while the amplitude of the rest of the states is zero. As such, there are exactly $\frac{M}{r}$ non-zero amplitudes. The superposition states will have normalizing factor of $\alpha_{nr} = \sqrt{\frac{r}{M}}$ such

that $\sum_{i=0}^{M} \alpha_i^2 = \frac{M}{r}\left(\sqrt{\frac{r}{M}}\right)^2 = 1$. Once the QFT is applied to $\alpha_i|\mathrm{j_i}\rangle$ the resulting state according to the result in the equation (4-6) is also periodic. The new period of the state is now $\frac{M}{r}$ with amplitude $\frac{1}{\sqrt{r}}$.

$$\sqrt{\frac{r}{M}}\sum_{i=0}^{\frac{M}{r}-1} \alpha_i|\mathrm{j_i}\rangle \rightarrow^{QFT} \frac{1}{\sqrt{M}}\sum_{k=0}^{M-1} \beta_k|\mathrm{j_k}\rangle \qquad \textbf{(4-6)}$$

$$= \begin{cases} \sum_{k=0}^{M-1} \sqrt{\frac{r}{M}}\frac{1}{\sqrt{M}}\omega_M^{jrk} = \frac{M}{r}\frac{\sqrt{r}}{M} = \frac{1}{\sqrt{r}}\ |\mathrm{j_k}\rangle, & k = \frac{nM}{r}, n\ in\ integer \\ 0, & k = 0 \end{cases}$$

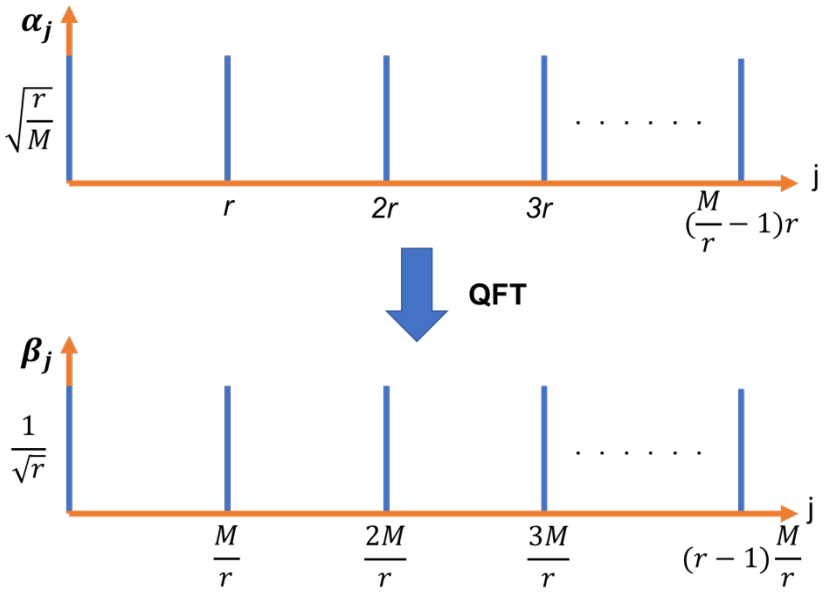

Figure 22: QFT Mapping for Periodic Superposition

The output state $\beta_k|\mathrm{j_k}\rangle$ has a contribution from all the states in $\alpha_i|\mathrm{j_i}\rangle$ which is multiplied by the phase term ω_N^{jrk}. The interesting fact is that due to constructive interference the new states have non-zero amplitude for integer multiples of $\frac{M}{r}$, for which the phase term has a value of one. In contrast, the remainder of the states decay to zero due to deconstructive interference. In this case, the summation of the phase term $\sum_{k=0}^{M-1} \omega_M^{jrk}$ is equal to zero. Measuring the output state $\beta_k|\mathrm{j_k}\rangle$, returns v, $n = 0, \ldots, r-1$ with equal probability of $\frac{1}{r}$. In order to determine r, the measurement is

repeated multiple times to obtain the set $\frac{n_1 M}{r}$, $\frac{n_2 M}{r}$, …, $\frac{n_d M}{r}$. Computing the pairwise GCD of the numbers in the set, $\frac{M}{r}$ is obtained. Since M is known the value of r can be determined.

5.5 QFT Circuit

Given the transformation, the next step is to drive the unitary quantum circuit elements required to implement QFT. The idea is to expand the transformation with regards to the basis states for each qubit in order to construct the gates. Assume $N = 2^n$, where n is an integer and the standard basis $|0\rangle, \dots, |2^{\mathrm{n}} - 1\rangle$ is used for an n qubit system. Note that the basis vectors were expressed as digits and can be converted to a binary sequence for the state $|\mathrm{y}\rangle = |\mathrm{y}_1 \mathrm{y}_2 \dots \mathrm{y}_{\mathrm{n}}\rangle = |\mathrm{y}_1 2^{\mathrm{n}-1} + \mathrm{y}_2 2^{\mathrm{n}-2} \dots \mathrm{y}_{\mathrm{n}} 2^0\rangle$.

$$QFT|\mathrm{x}\rangle = \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} \omega_N^{xy} |\mathrm{y}\rangle, \quad \text{definition of the QFT} \tag{4-7}$$

$$= \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{i2\pi xy/2^n} |\mathrm{y}\rangle, \tag{4-8}$$

➔ where $\omega_N^{xy} = e^{i2\pi xy/N}$ and $N = 2^n$

$$= \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{\frac{i2\pi xy}{2^n}} |\mathrm{y}_1 y_2 \dots y_{n-1} y_n\rangle, \tag{4-9}$$

➔ $binary\ representation$ $|\mathrm{y}\rangle = |\mathrm{y}_1 y_2 \dots y_{n-1} y_n\rangle$

$$= \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{i2\pi x\left(\sum_{k=0}^{n} \frac{y_k}{2^k}\right)} |\mathrm{y}_1 y_2 \dots y_{n-1} y_n\rangle, \tag{4-10}$$

➔ fractional binary notation $\frac{y}{2^n} = \sum_{k=0}^{n} \frac{y_k}{2^k}$

$$= \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} \prod_{k=0}^{n} e^{i2\pi x \frac{y_k}{2^k}} |\mathrm{y}_1 y_2 \dots y_{n-1} y_n\rangle, \tag{4-11}$$

➔ exponential of sum = multiplication of exponential

$$= \frac{1}{\sqrt{N}} \bigotimes_{k=1}^{n} (|0\rangle + e^{2\pi i x/2^k} |1\rangle), \tag{4-12}$$

➔ tensor product of each qubit state

$$= \frac{\left(|0\rangle + e^{2\pi i[0.x_n]}|1\rangle\right)\left(|0\rangle + e^{2\pi i[0.x_{n-1}x_n]}|1\rangle\right) \dots \left(|0\rangle + e^{2\pi i[0.x_1 x_2 \dots x_{n-1}x_n]}|1\rangle\right)}{2^{\frac{n}{2}}} \tag{4-13}$$

$where\ \ 0.x_1 x_2 \dots x_{n-1} x_n\ \ is\ the\ binary\ fraction\ notation\ \frac{x_1}{2} + \frac{x_2}{4} + \cdots \frac{x_{n-1}}{2^{n-1}} + \frac{x_n}{2^n}$

The product representation obtained at the end of the derivation in (4-13), greatly simplifies the implementation of QFT. The derivation decomposes the circuit into a Hadamard gate operation and a Controlled Phase gate which is applied iteratively. A Controlled Phase gate is a unitary operator constructed by following the general structure of controlled gates examined previously.

$$\mathrm{CU}(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\theta} \end{bmatrix} \tag{4-15}$$

Considering the product term, note that the last qubit y_n depends on the all input qubit values from $x_1x_2 \ldots x_{n-1}x_n$. Moving from the last output qubit to the first one, there is a clear trend that qubits depend less and less on the input qubits. To gain an intuitive understanding of the operation on each qubit, the QFT of a 3-qubit system is inspected and the circuit is constructed.

Given $n = 3, N = 2^n = 8$

$$|y_1y_2y_3\rangle = QFT|x_1x_2x_3\rangle V \tag{4-16}$$
$$= \frac{1}{\sqrt{8}}\left(|0\rangle + e^{2\pi i[0.x_3]}|1\rangle\right)\left(|0\rangle + e^{2\pi i[0.x_2x_3]}|1\rangle\right)\ldots\left(|0\rangle + e^{2\pi i[0.x_1x_2x_3]}|1\rangle\right)$$

Step 1- Apply a Hadamard gate to $|x_3\rangle$ to get $|0\rangle + e^{2\pi i[0.x_3]}|1\rangle = |0\rangle + e^{2\pi i x_3/2}|1\rangle = |0\rangle + (-1)^{x_3}|1\rangle$. Note this expression captures the two possible scenarios, when a Hadamard gate is applied to a pure state $|x_3\rangle$ which can either be in $|0\rangle$ or $|1\rangle$ state.

$$H|0\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle = |0\rangle + (-1)^0|1\rangle \tag{4-17}$$

$$H|1\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle = |0\rangle + (-1)^1|1\rangle \tag{4-18}$$

Step 2- Apply Hadamard gate to $|x_2\rangle$ and $CU\left(\frac{\pi}{2}\right)$ with the target on $|x_3\rangle$, giving the state $|0\rangle + e^{2\pi i[0.x_2x_3]}|1\rangle$.

Step 3- Apply Hadamard gate to $|x_3\rangle$ and $CU\left(\frac{\pi}{2}\right)$ with the target on $|x_2\rangle$, and another $CU\left(\frac{\pi}{4}\right)$ with the target on $|x_3\rangle$, giving the state $|0\rangle + e^{2\pi i[0.x_2x_3]}|1\rangle$.

Step 4- Finally the qubits are measured in the reverse order where $y_1 = x_3, y_2 = x_2\ y_1 = x_3$.

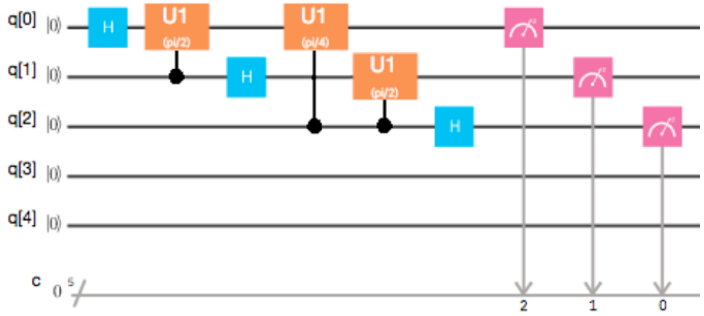

Figure 23: QFT circuit on IBM's Quantum Experience [29]

The circuit in figure 23 shows the QFT realization on IBM's Quantum Experience platform. Given the QFT circuit, the period finding problem can be solved. Mathematically the goal is finding r such that $F(r) = a^r \ mod\ N = 1$. As such, the final requirement is to construct a circuit to implement $F(x)$ the modular exponential function. Then by utilizing constructional interference of amplitudes the period can be measured.

5.6 Modular Exponentiation Circuit

On a classical computer, the function can only be evaluated at individual points, as a result finding the period (a global property) is a very resource intensive and time-consuming task.

The required mapping for this gate is:

$$U(a): |0\rangle \longmapsto a^x\ (mod\ N) \qquad \textbf{(4-19)}$$

This modular exponential circuit is very challenging to construct, as a general algorithm is needed to consider a as a parameter and form the circuit accordingly. This is the main obstacle preventing full implementation of Shor's algorithm for practical values of N [22]. Many circuits have been proposed, however they are very resource intensive. The huge number of gates required, cannot be realized using the relatively small quantum computers available today [23]. Consequently,

many researchers have shifted to compiled circuits which implement the function $U(a)$ for specific values of a. To simplify the process even further, a small of value N is chosen along with an appropriate value. Afterwards a truth table is constructed for the function $U(a)$ and the circuit is created by inspection. Figure 24 shows an example of a compiled circuit for the modular exponential function pre-determined for values of $N = 15\ and\ a = 2$.

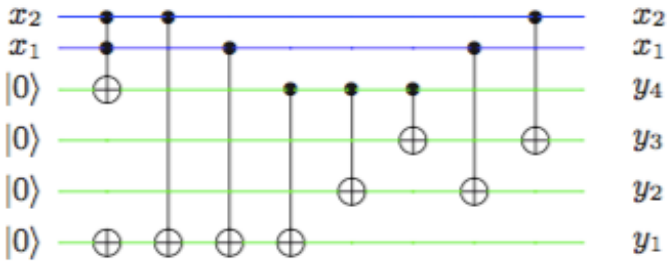

Figure 24: Modular Exponential for N=15, a=2 [22]

Some researchers have cautioned that implementing the compiled version of Shor's algorithm is not a genuine demonstration of solving the factoring problem [24]. However, the compiled circuits present an opportunity to model Shor's algorithm on current prototype quantum computers. This permits the extraction of valuable data to characterize the performance of the algorithm.

5.7 Shor's Implementation on IBM Q

A compiled version of the circuit was tested on IBM Quantum Experience platform for factoring the number N=15 with chosen a=7. The circuit was tested on QASM local simulators and IBM QX5 16 qubit quantum processor. The full circuit is illustrated in figure 25. The numbered boxes correspond to the following list of steps in the quantum subroutine:

1. Inducting superposition state with Hadamard Gate.
2. Calculating the modular exponentiation using the compiled circuit for N=15 and a=7.
3. The inverse QFT applied to the superposition state.
4. Measurement is taken to extract the periodicity of the modular exponential function.

The results of the circuit on IBM Quantum simulators are plotted in figure 26 with 1024 shots. The shot parameter specifies the number of times the quantum program was executed. As presented by the graph the final state is in mixture of states 000, 010, 100, 110 corresponding to values of 0, 4, 2, 6, since the MSB is the right-most bit. Recall that the result of the QFT is an

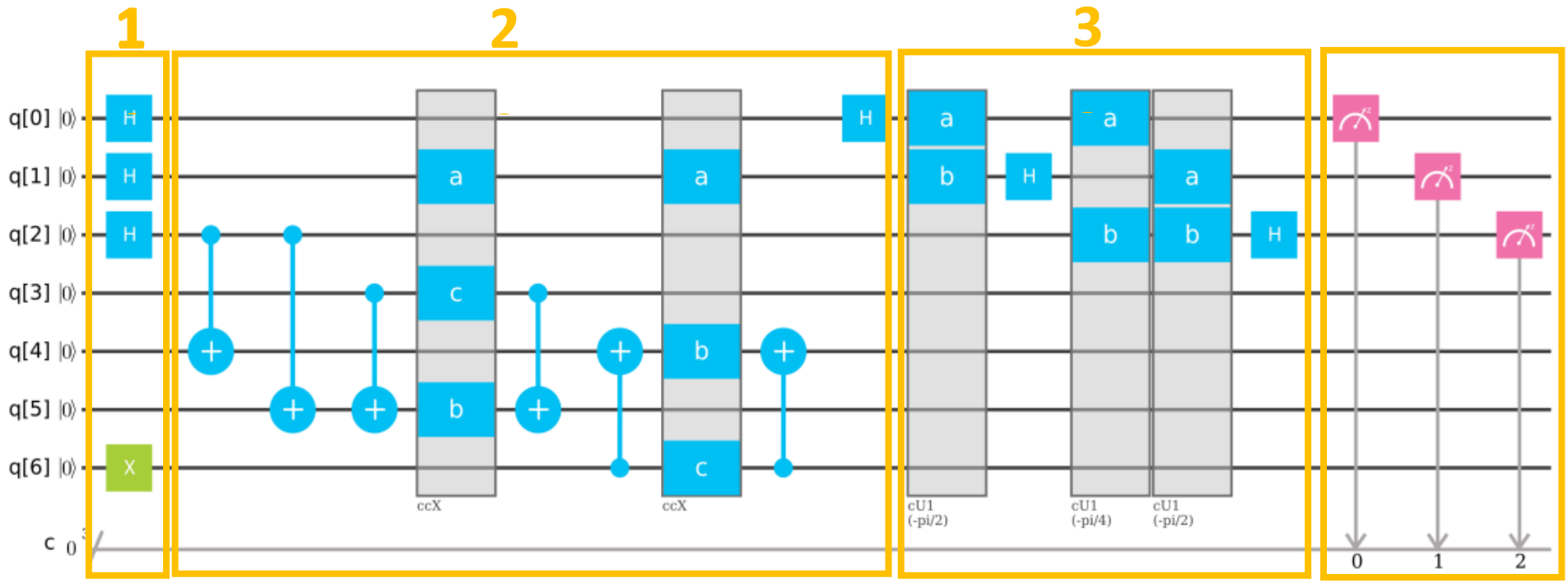

Figure 25: Shor's circuit implementation on IBM Quantum Experience for factoring N=15

Figure 26: Shor's circuit results ran on IBM Quantum simulators with 1024 shots

integer multiple of $\frac{M}{r}$, where M is the total number of states. For the purposes of this experiment with 3 qubits M is equivalent to $2^3 = 8$. Next the pairwise greatest common divisor of the set 0, 2, 4, 6 is computed. The value 0 is the failure state of the problem which can be simply ignored. It is easy to validate that the GCD of the set is 2, equivalent to $\frac{8}{r}$. As such the value of r is determined to be $r = \frac{8}{2} = 4$. Proceeding to the last step in the algorithm the factors are discovered by plugging in the period into the following formulas.

$$p = \gcd\left(a^{r/2} + 1, N\right) = \gcd\left(7^{4/2} + 1, 15\right) = \gcd(50, 15) = 5$$

$$q = gcd(a^{r/2} - 1, N) = \gcd\left(7^{\frac{4}{2}} - 1, 15\right) = \gcd(48, 15) = 3$$

Figure 27 presents the results of the execution of the circuit in figure 25 on IBM QX5 hardware. Note that the bits strings shown in this graph follow the convention where the MSB is the left-most-bit. The results obtained from the hardware contain four extra garbage states, produced due to the decoherence of the qubits and gate errors. The top four states match the simulation results. This is due to the fact that simulation does not consider the error present in the system. The probability of the garbage states can be slightly reduced by increasing the number of shots used. A greater improvement can possibly be achieved by employing an error correcting algorithm.

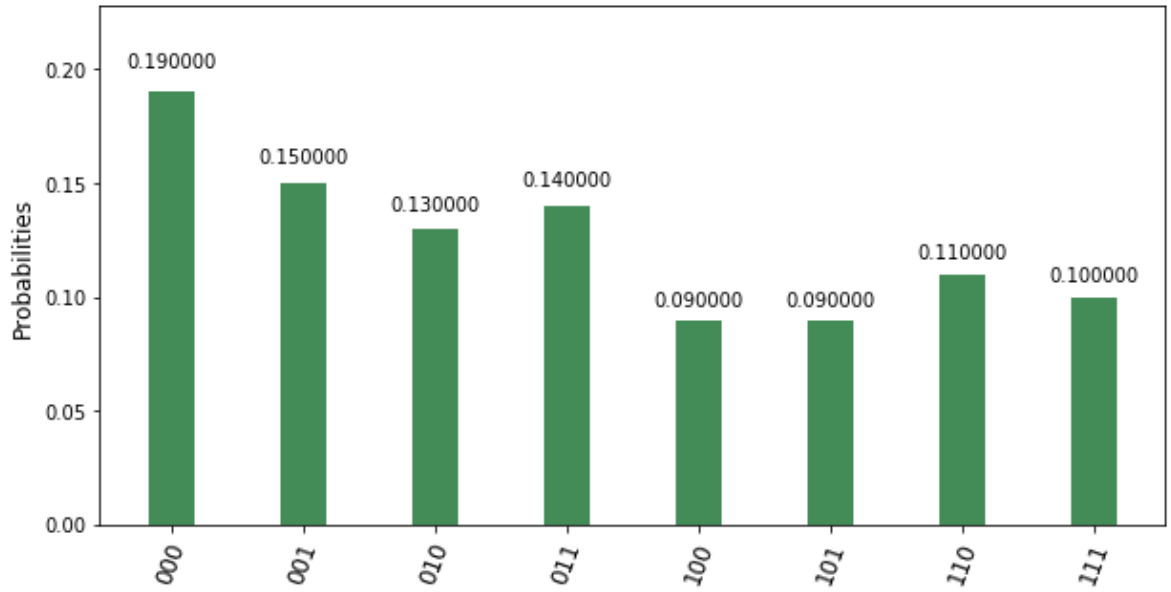

Figure 27: Shor's circuit results ran on IBMQX5, 16 qubit processor with 1024 shots

5.8 Shor's Complete Simulation on Project Q

There are many different proposed implementations of Shor's algorithm. Some implementations focus on optimizing the number of qubits, while others minimize the depth of the circuit. The realization used for this experiment was based on a paper by Beauregard published in 2003 [25]. Table 3 presents the results from this experiment. N corresponds to the number being factored and n is the log of N equivalent to the number of qubits used to store N. The prime factors of N are p and q. The constant a is randomly chosen for the modular exponentiation function and r is the periodicity of the function. The max-width column presents the maximum number qubits used at any single time during the computation. The proceeding columns are the various gates employed in the algorithm. The significance of the experiment is the large number of gates that is needed for factoring relatively small integers displayed under the total number of gate column. The maximum number of gates allowed on IBM quantum platform is currently 1000. Contrary to the common belief that the number of qubits is the limiting factor in realization of many quantum algorithms, it is clear from the experiment that circuit depth is also a key factor. The error in the quantum system builds up over time and many operations. As such, the state of the system will be lost after a certain depth depending on the decoherence timescales and gate error values. Reducing the complexity of resources required on a quantum computer for executing Shor's algorithm is an active area of research. Modular exponentiation is the bottleneck of this quantum subroutine. The state of art realization of this function still requires $O(n^2)$ gates.

N	n	p	q	a	r	Max-width	H Gate	X Gate	CX Gate	CSWAP Gate	CR Gate	R Gate	Measure	Total Number of Gates
9	4	3	3	5	2	11	4240	394	384	96	11872	967	27	17980
15	4	5	3	8	4	11	5648	525	512	128	15850	1288	36	23987
21	5	3	7	2	6	13	13464	1138	1112	278	41763	3105	69	60929
33	6	11	3	7	10	15	17788	1434	1400	350	58305	4124	82	83483
39	6	3	13	32	2	15	26468	2019	1976	494	91811	6151	108	129027
51	6	3	17	20	2	15	30788	2308	2264	566	108397	7159	121	151603
57	6	3	19	47	18	15	35116	2603	2552	638	125032	8178	134	174253
69	7	3	23	8	22	17	41796	3005	2944	736	154184	9759	149	212573
87	7	3	29	83	14	17	48480	3403	3336	834	183396	11340	164	250953
93	7	3	31	68	2	17	61822	4193	4120	1030	241943	14487	194	327789
111	7	3	37	56	18	17	68492	4593	4512	1128	271172	16068	209	366174
123	7	3	41	79	2	17	75162	4986	4904	1226	300115	17636	224	404253
129	8	3	43	110	42	19	94616	6032	5928	1482	394883	22272	258	525471
141	8	3	47	104	234	19	114114	7073	6952	1738	490201	26908	292	647278
159	8	3	53	155	26	19	133596	8113	7976	1994	585450	31544	326	768999
183	8	3	61	46	10	19	172540	10202	10024	2506	775534	40817	394	1012017
201	8	3	67	194	22	19	192028	11244	11048	2762	870591	45454	428	1133555
213	8	3	71	95	70	19	211538	12284	12072	3018	965768	50090	462	1255232
237	8	3	79	32	26	19	231016	13326	13096	3274	1060879	54726	496	1376813
249	8	3	83	50	242	19	240766	13849	13608	3402	1108491	57045	513	1437674
267	9	3	89	46	4	21	254386	14500	14256	3564	1181787	60286	532	1529311
291	9	97	3	208	96	21	295308	16474	16200	4050	1402975	70056	589	1805652
485	9	97	5	83	96	21	308918	17134	16848	4212	1476444	73313	608	1897477
679	10	97	7	261	32	23	327328	17940	17648	4412	1585095	77717	629	2030769
1067	11	97	11	41	480	25	351562	18914	18616	4654	1740398	83544	652	2218340
1261	11	13	97	751	24	25	375810	19895	19584	4896	1895756	89373	675	2405989

Table 3 Result of factoring sample numbers on Project Q simulators using Shor's implementation by Beauregard (2003) [25]

5.9 Quantum Attack on RSA

The significance of Shor's algorithm is the threat it poses for public key cryptography algorithms such as RSA. In theory the exponential speedup provided by Shor's subroutine is capable of breaking RSA key in reasonable amount of time. Consequently, it is of interest to assess the resources required for such an attack by quantum computers. Table 4 lists the qubit and depth complexity of two proposed implementations for Shor's algorithm. The number qubits required to break key size of 2048 bits is in the order of thousands while the depth required is close to 8 billion. The complexity of these resources is far beyond the capability of today's best quantum computers.

Proposed Implementation of Shor's	**Beauregard (2003)** [25]	**Pavilidis et al. (2012)** [26]
Qubits Complexity	2n+3	9n+2
*Depth Complexity	$O(n^3)$	$2000n^2$
Qubits required for RSA 2048	4099	18,434
*Depth required for RSA 2048	8,589,934,592	8,388,608,000

Table 4 Complexity comparison of two proposed implementations of Shor's algorithm

The development of quantum computing hardware is still in its infancy stages, where various architectures are being explored. Each platform has its own strengths and weaknesses in computation power. There are many different factors that determine the computation power of these processors. Table 5 lists these factors and their current state on IBM Q processors. The number of qubits is main metric that is often publicized and valued higher than the other factors. However, as the results in section 5.8 indicated, the depth of circuit can be an equally important metric.

Factors affecting computation power of quantum computer	**Current state of the quantum architecture**
Number of Qubits	O (100)
Depth of Circuit	O (1000)
Connectivity of Qubits	Limited Connectivity (to 2 neighbors)
Number of operations that can be run in parallel	Limited Parallelism

Table 5 Analysis of factors affecting computation power of quantum computers

In order to fairly compare between technologies and track the progress quantum processors, IBM researchers proposed technology neutral metric known as quantum volume. It is the product of the number of qubits and maximum number of operations that can be performed before the system

behaves classically. This will produce a single value which can easily be compared across platforms.

6. Conclusion

Over two decades ago, two of today's most discussed quantum algorithms, Shor's and Grover's were introduced. Since their discovery, it was realized that they both provided a great computation speedup in theory compared to their classical equivalents. This fact has posed a great threat to the future of many public key cryptography routines based on integer factorization, discrete log and elliptic curves. Given a powerful universal quantum computer the security of many public key cryptography algorithms is significantly weakened. In order to obtain beneficial information regarding the impact of quantum algorithms on cryptography routines, compiled circuits for Shor's and Grover's were executed on available hardware.

Currently the full implementation of Shor's cannot be implemented on hardware. Optimized modular exponentiation circuits need to be designed along with more powerful quantum computers. An experimental analysis of the algorithm was carried out in simulation, which highlighted the significant number of gates required to carry out Shor's algorithm. RSA is widely used public key routine which is greatly threatened by the arrival quantum computers. NIST currently recommends a key size of 2048 bit for RSA. It was determined that thousands of qubits along with billions of gates are required to break this key size on quantum computer. As such the current capabilities of quantum computers is about two orders of magnitude less than the requirements for Shor's algorithm.

The development of quantum hardware has progressed greatly over the recent years. Various small-scale quantum processes have been built and many are accessible through the cloud. Many different architectures have been proposed and are under investigation. It is important to recognize that the number of qubits, is not the sole metric for quantum computation power. The effective error rate of the system is also essential which limits the depth of the circuit which can be realized on hardware. Consequently, universal metrics such as quantum volume should be considered to compare different architectures and benchmark progress for specific algorithms.

As an area of future research, the impact of Grover's algorithm on lattice-based cryptography can be explored. An oracle function has been proposed for solving the Shortest Vector Problem (SVP), which can be studied and simulated.

7. Bibliography

[1] D. Castelvecchi, "Quantum computers ready to leap out of the lab in 2017," 03 January 2017. [Online]. Available: http://www.nature.com/news/quantum-computers-ready-to-leap-out-of-the-lab-in-2017-1.21239. [Accessed 05 10 2017].

[2] T. Watson, "IBM doubles compute power for quantum systems, developers execute 300K+ experiments on IBM Quantum Cloud," IBM, 17 May 2017. [Online]. Available: https://developer.ibm.com/dwblog/2017/quantum-computing-16-qubit-processor/. [Accessed 05 October 2017].

[3] J. M. Gambetta, J. M. Chow and M. Steffen, "Building logical qubits in a superconducting quantum computing system," *Quantum Information 3, Article number: 2,* 2017.

[4] S. Boixo, S. V. Isakov, Vadim N. Smelyanskiy, R. Babbush, N. Ding, Z. Jiang, M. J. Bremner, J. M. Martinis and H. Neven, "Characterizing Quantum Supremacy in Near-Term Devices," *arXiv:1608.00263v3 [quant-ph],* 5 Apr 2017.

[5] J. Bloomberg, "This Is Why Quantum Computing Is More Dangerous Than You Realize," Forbes, 11 August 2017. [Online]. Available: https://www.forbes.com/sites/jasonbloomberg/2017/08/11/this-is-why-quantum-computing-is-more-dangerous-than-you-realize/#7ca92ffe3bab. [Accessed 9 October 2017].

[6] P. W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," *arXiv:quant-ph/9508027,* 1996.

[7] C. Figgatt, D. Maslov, K. A. Landsman, N. M. Linke, S. Debnath and C. Monroe, "Complete 3-Qubit Grover Search on a Programmable Quantum ComputerC. Figgatt,1 D. Maslov,2, 1 K. A. Landsman,1 N. M. Linke,1 S. Debnath,1 and C. Monroe1, 3," *arXiv:1703.10535,* 2017.

[8] E. Gerjuoy, "Shor's Factoring Algorithm and Modern Cryptography. An Illustration of the Capabilities Inherent in Quantum Computers," *arXiv:quant-ph/0411184,* 2004.

[9] R. O'Donnell, "Quantum Computation and Information," [Online]. Available: https://www.cs.cmu.edu/~odonnell/quantum15/.

[10] R. P. Feynman, "Simulating Physics wth Computers," *International Journal of Theoretical Physics,* vol. 21, no. 6/7, p. 467–488, 1982.

[11] S. Jordan, "Quantum Algorithm Zoo," 18 January 2018. [Online]. Available: https://math.nist.gov/quantum/zoo/. [Accessed 25 January 2018].

[12] L. Bishop, "QISKit and Quantum Computing," 29 March 2017. [Online]. Available: https://developer.ibm.com/open/videos/qiskit-quantum-computing-tech-talk/.

[13] M. Nielsen and I. Chuang, Quantum Computation and Quantum Information, New York: Cambridge University Press, 2010.

[14] R. Haughton, "Race for quantum supremacy hits theoretical quagmire," *Nature,* 2017.

[15] N. M. Linke, D. Maslov, M. Roetteler, S. Debnath, C. Figgatt, K. A. Landsman, K. Wright and C. Monroe, "Experimental Comparison of Two Quantum Computing Architectures".

[16] "Welcome to the IBM Q Experience," [Online]. Available: https://quantumexperience.ng.bluemix.net/qx/experience.

[17] L. Grover, "A fast quantum mechanical algorithm for database search," *arXiv:quant-ph/9605043,* 1996.

[18] J. Watrous, "Lecture 12: Grover's Algorithm," University of Calgary, 7 March 2006. [Online]. Available: https://cs.uwaterloo.ca/~watrous/LectureNotes/CPSC519.Winter2006/12.pdf.

[19] A. Dawar, "Quantum Computing," University of Cambridge, 2009. [Online]. Available: https://www.cl.cam.ac.uk/teaching/0910/QuantComp/.

[20] D. Kemp, "Animation of Grover's Quantum Search Algorithm," [Online]. Available: http://davidbkemp.github.io/animated-qubits/grover.html.

[21] T. Kleinjung, K. Aoki, J. Franke, A. Lenstra, E. Thomé, J. Bos, P. Gaudry, A. Kruppa, P. Montgomery, D. A. Osvik, H. t. Riele, A. Timofeev and P. Zimmermann, "Factorization of a 768-bit RSA modulus," *Cryptology ePrint Archive,* vol. 6, 2010.

[22] O. Gamel and D. James, "Simplified Factoring Algorithms for Validating Small-Scale Quantum Information Processing Technologies," *arXiv:1310.6446,* 2013.

[23] R. Meter and K. Itoh, "Fast Quantum Modular Exponentiation," *arXiv:quant-ph/0408006,* vol. 71, no. 5, 2005.

[24] J. Smolin, G. Smith and A. Var, "Pretending to factor large numbers on a quantum computer," *arXiv:1301.7007,* 2013.

[25] S. Beauregard, "Circuit for Shor's algorithm using 2n+3 qubits," *Quantum Information and Computation,* vol. 3, no. 2, pp. 175-185, 2003.

[26] P. Archimedes and G. Dimitris, "Fast Quantum Modular Exponentiation Architecture for Shor's Factorization Algorithm," *Quantum Information and Computation,* vol. 14, no. 7, p. 34, 2013.

[27] I. Chuang, R. Laflamme, P. Shor and W. Zurek, "Quantum Computers, Factoring, and Decoherence," *arXiv:quant-ph/9503007,* 1995.

[28] L. Bishop, "IBM Q Experience Documentation," 2017. [Online]. Available: https://quantumexperience.ng.bluemix.net/qx/tutorial?sectionId=full-user-guide&page=002-The_Weird_and_Wonderful_World_of_the_Qubit~2F035-Decoherence.

[29] A. Phan, " Quantum Fourier Transform," IBM, 2017. [Online]. Available: https://github.com/QISKit/qiskit-tutorial/blob/master/2_quantum_information/fourier_transform.ipynb.

Appendix A Shor's Circuit QASM Code

```
import sys
if sys.version_info < (3, 5):
    raise Exception('Please use Python version 3.5 or greater.')
from qiskit import QuantumProgram
import Qconfig
qp = QuantumProgram()
# Creating Registers
# create your first Quantum Register called "qr" with 2 qubits
qr = qp.create_quantum_register('qr', 7)
# create your first Classical Register  called "cr" with 2 bits
cr = qp.create_classical_register('cr', 3)
# Creating Circuits
# create your first Quantum Circuit called "qc" involving your Quantum Register
"qr"
# and your Classical Register "cr"
qc = qp.create_circuit('Circuit', [qr], [cr])
# get the circuit by Name
circuit = qp.get_circuit('Circuit')
# get the Quantum Register by Name
q = qp.get_quantum_register('qr')
# get the Classical Register by Name
c = qp.get_classical_register('cr')
# Shor's Circuit for factoring number N=15 with a=2
circuit.h (q[0]);
circuit.h (q[1]);
circuit.h (q[2]);
circuit.x (q[6]);
circuit.cx (q[2],q[4]);
circuit.cx (q[2],q[5]);
circuit.cx (q[3],q[5]);
circuit.ccx (q[1],q[5],q[3]);
circuit.cx (q[3],q[5]);
circuit.cx (q[6],q[4]);
circuit.ccx (q[1],q[4],q[6]);
circuit.cx (q[6],q[4]);
circuit.h (q[0]);
circuit.cu1((-pi/2), q[0],q[1]);
circuit.h (q[1]);
circuit.cu1((-pi/4), q[0],q[2]);
circuit.cu1((-pi/2), q[1],q[2]);
circuit.h (q[2]);
circuit.barrier(q)
circuit.measure(q[0], c[0])
circuit.measure(q[1], c[1])
circuit.measure(q[2], c[2])
backend = 'ibmqx5'  # Backend where you execute your program; in this case, on the
real Quantum Chip online
circuits = ['Circuit']   # Group of circuits to execute
shots = 1024           # Number of shots to run the program (experiment); maximum
is 8192 shots.
max_credits = 10          # Maximum number of credits to spend on executions.
qobj=qp.compile(circuits, backend)
result_real = qp.run(qobj, wait=2, timeout=2
```

Appendix B Grover's Circuit QASM Code

```
import sys

if sys.version_info < (3, 5):
    raise Exception('Please use Python version 3.5 or greater.')

from qiskit import QuantumProgram
import Qconfig
import matplotlib.pyplot as plt
% matplotlib
inline
import numpy as np
from scipy import linalg as la
from qiskit.tools.visualization import plot_histogram, plot_state

qp = QuantumProgram()
qp.enable_logs()
# Creating Registers
# create your first Quantum Register called "qr" with 2 qubits
qr = qp.create_quantum_register('qr', 5)
# create your first Classical Register  called "cr" with 2 bits
cr = qp.create_classical_register('cr', 5)
# Creating Circuits
# create your first Quantum Circuit called "qc" involving your Quantum Register
"qr"
# and your Classical Register "cr"
qc = qp.create_circuit('Circuit', [qr], [cr])
# get the circuit by Name
circuit = qp.get_circuit('Circuit')
# get the Quantum Register by Name
q = qp.get_quantum_register('qr')
# get the Classical Register by Name
c = qp.get_classical_register('cr')
circuit.x(q[4])
circuit.h(q[0])
circuit.h(q[1])
circuit.h(q[3])
circuit.h(q[4])
circuit.x(q[1])
circuit.ccx(q[0], q[1], q[2])
circuit.ccx(q[2], q[3], q[4])
circuit.ccx(q[0], q[1], q[2])
circuit.ccx(q[2], q[3], q[4])
circuit.x(q[1])
circuit.h(q[0])
circuit.h(q[1])
circuit.h(q[3])
circuit.x(q[0])
circuit.x(q[1])
circuit.x(q[3])
circuit.h(q[3])
circuit.ccx(q[0], q[1], q[3])
circuit.h(q[3])
circuit.x(q[0])
circuit.x(q[1])
circuit.x(q[3])
circuit.h(q[0])
```

```
circuit.h(q[1])
circuit.h(q[3])
circuit.h(q[4])
circuit.measure(q[0], c[0])
circuit.measure(q[1], c[1])
circuit.measure(q[3], c[3])
circuit.measure(q[4], c[4])
backend = 'ibmqx5'  # Backend where you execute your program; in this case, on the
real Quantum Chip online
circuits = ['Circuit']  # Group of circuits to execute
shots = 1024  # Number of shots to run the program (experiment); maximum is 8192
shots.
max_credits = 10  # Maximum number of credits to spend on executions.
coupling_map = {
    1: [0, 2], 2: [3], 3: [4, 14], 5: [4], 6: [5, 7, 11], 7: [10], 8: [7], 9: [8,
10], 11: [10], 12: [5, 11, 13],
    13: [4, 14], 15: [0, 2, 14]
}
initial_layout = {
    ("q", 0): ("q", 0), ("q", 1): ("q", 1), ("q", 2): ("q", 2), ("q", 3): ("q", 3),
("q", 4): ("q", 4),
    ("q", 5): ("q", 5), ("q", 6): ("q", 6), ("q", 7): ("q", 7)
}
qobj = qp.compile(circuits, backend, coupling_map=coupling_map)
result_real = qp.run(qobj, wait=2, timeout=2400)
ran_qasm = result.get_ran_qasm('Circuit')
# result_real = qp.execute(circuits, backend=backend, shots=shots,
max_credits=max_credits, wait=10, timeout=2040)
result_real.get_counts('Circuit')
```