# Quantum Computation

Quantum Computing Project 2017/2018


Agnibho Chattarji s1552184
Fidel Elie s1510784
Ewan Morrison s1539149
Alasdair Reid s1539349
Jake Smith s1415942



**Abstract**

The cutting edge field of quantum computing may well result in a revolution in information processing and data science, with many classically complex problems having significantly faster solutions due to quantum algorithms. The following report provides a basic introduction to quantum computing, assuming some previous knowledge of quantum mechanics and linear algebra. Two quantum algorithms, Grover's Algorithm and Shor's Algorithm, are examined, and a classical simulation of a quantum computer running Grover's algorithm is presented and discussed. Beyond this, potential methods for physically implementing such computers are summarised, along side the applications and implications of this rapidly developing field.


Date Performed: January, 2018 - March, 2018
Course Organiser: Anthony Kennedy

# Contents

# 1 Introduction

## 1.1 Motivation

The progress of computers over the last 60 years has been a seemingly unending series of amazing advancements, with each new generation of technology pushing the boundaries of what seemed possible. A modern smartphone has more computing power than the computers that put a man on the moon, more so even than a room sized military computer of 50 years ago. This progress has been achieved by making transistors (the basic switching and memory units of computers) orders of magnitude smaller than those that came before. In 1988 the Intel i960 chip had 180,000 transistors, but just one year later in 1989 the Intel 80486 chip counted 1,180,235 transistors. Seeing the progress of computing power, in 1965 Gordon Moore made the prediction that the density of transistors would double roughly every 18 months, known as Moore's law. Figure 1 below shows how this law has been holding true for over 40 years.

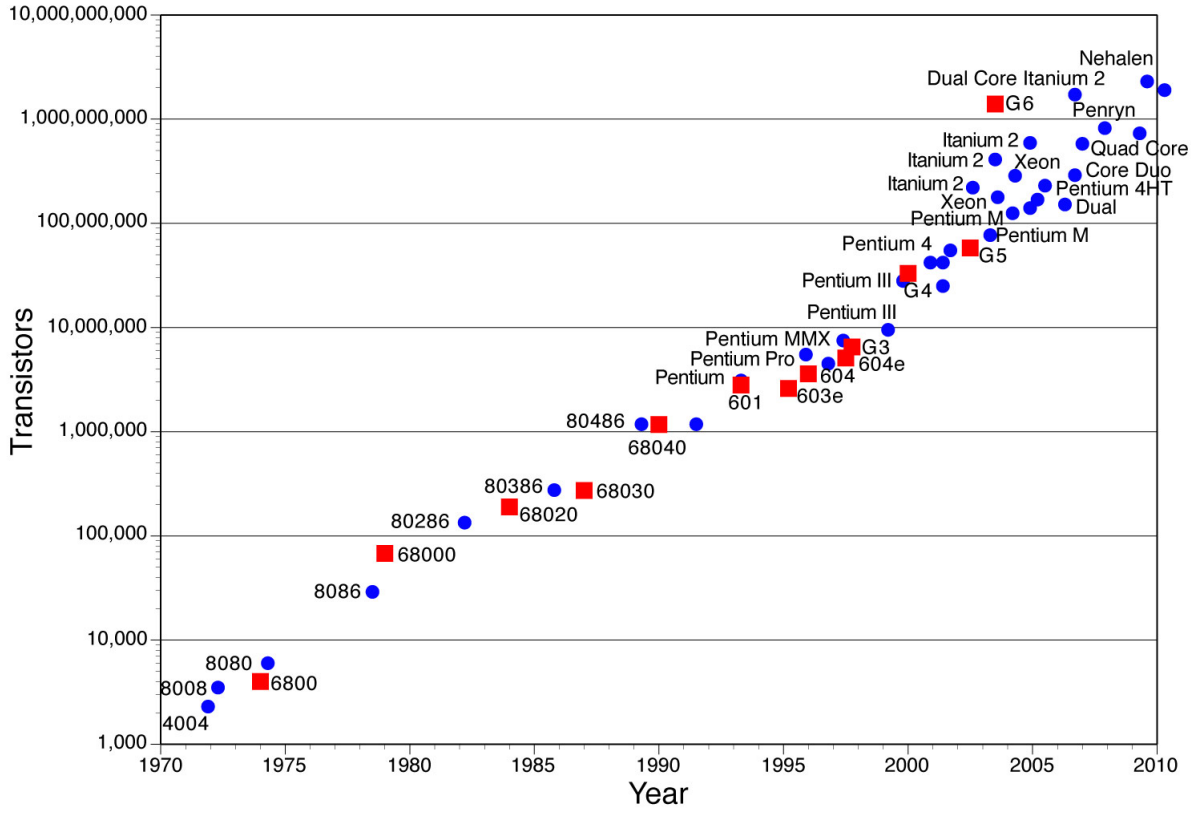


Figure 1: A graph demonstrating Moore's law[1].

However, this trend cannot continue indefinitely, as there will come a time when transistors approach the size of individual atoms and will become affected by the laws of quantum mechanics. There are even some estimates that this point will be reached within the next 6 years[2]. Once quantum mechanics comes into play, its laws will then define how the transistors act, meaning that computing will have to be done in a radically different way in order to keep progressing. If, however, a computer can be built that exploits quantum effects, a *quantum computer*, it will have the potential to be millions of times faster than computers in use today.

This paper will introduce quantum computers; starting with an introduction to classical and quantum computing, moving into a more technical discussion introducing qubits and quantum gates. Building on this, the paper will introduce Grover's algorithm, then describe a computer programme built to implement this algorithm, namely its design and success[3–6].

## 1.2 Classical Computing

The classical computer is based in implementing two key abilities: the ability to store numbers in memory; and the ability to processes those stored numbers through mathematical operations. By combining a string of simple mathematical operations such as addition and subtraction, known as an *algorithm*, a computer has the ability to perform more complex operations. Both storage and processing are achieved through the use of transistors. A transistor acts by existing in two states, on or off, representing a binary value of 1 or 0 respectively. Through the binary language large strings of these transistors can store large amounts of data. A USB stick with the ability to store 512 megabytes of data is capable of doing so because it contains over 4 billion transistors.

Computers perform calculations through the use of circuits known as logic gates. Combinations of OR, AND, NOT, and other such gates are made through connecting transistors together. Computers perform calculations by comparing patterns of bits held in *registers* (temporary memory) and turn them into a new pattern of bits. This change may represent a process such as addition or subtraction. In physical terms it is the output of one logic gate flowing into the input of another logic gate on a circuit board. On a large scale this process is how the classical computers of the world are able to do all that they do. The size of transistors will eventually reach a quantum scale however and to keep progress moving quantum computers will have to be created.

## 1.3 Quantum Computing

The question of transistors that take advantage of the laws of quantum mechanics has been considered for many decades. Among the first to examined the issue were the IBM research physicists Rolf Landauer and Charles Bennett. In the 1960s Landauer first proposed that information itself was a physical entity that could be manipulated through the laws of physics, and that computers waste energy by manipulating the bits inside them[7]. Following this work, in the 1970s Bennet proposed a way in which computers could circumvent this wasted energy by working in a way that is reversible, suggesting that a quantum computer could perform extremely expensive computations without the large amounts of wasted energy. In 1982 Richard Feynman sketched out a design for how a machine using principles based in quantum mechanics could carry out basic computations[8]. Building on this, a few years later David Deutsch of Oxford University proposed a more detailed theoretical basis for a quantum computer[9].

While the classical computer is built upon logic gates implemented by series of bits[10], a quantum computer would be built using quantum bits known as qubits. Where a bit has the ability to store either a one or a zero, a qubit has the ability to store a one, a zero, both one and zero and also an infinite number of values in between zero and one. Not only this but a qubit has the ability to be in multiple states at the same time, and so storing multiple values. Qubits are able to store multiple values by being in a superposition of multiple states simultaneously, much in the same way that Schrodinger's cat is in a superposition of being both dead and alive. As the qubit can occupy multiple states at the same time it also has the ability to to perform multiple processes in parallel, where as a classical computer must work through processes in series. The qubit can occupy multiple states until it is observed or measured, since the processes of measuring will cause the qubit to collapse into one of its possible states, with each possible state having some probability of observation attached to it. It has been estimated that a computer taking advantage of multiple states processed in parallel could be millions of times faster than a classical computer.

## 1.4 Physical Implementation of Quantum Computers

In practice, qubits would have to be atoms, ions, electrons or photons, and manipulation of their states could be achieved using lasers or electromagnetic fields, to name but a few. One example of this is a quantum simulator created at the University of Maryland: researchers were able to create a simulator made up of 53 qubits[11]. This simulator uses ytterbium ions held together in close proximity, using electric fields to hold them in a single neat row. Quantum magnetism is simulated in a vacuum chamber by manipulating the qubits with lasers. These lasers replicate quantum interactions between the particles. The manipulation by laser is done by making each qubit act like atomic magnets, where each ion has a north and south pole. By changing the strength of the lasers the researchers were able to achieve a desired state corresponding to a set of magnetic alignments. A physicist at the University of Maryland has claimed that the 53 qubit computer can simulate over a quadrillion different magnetic configurations, and that this number will double with each qubit that is added to the system[11]. With increasingly larger qubit systems being created, quantum simulators will be able to simulate far more complex atomic interactions that are far beyond that ability of current conventional supercomputers.

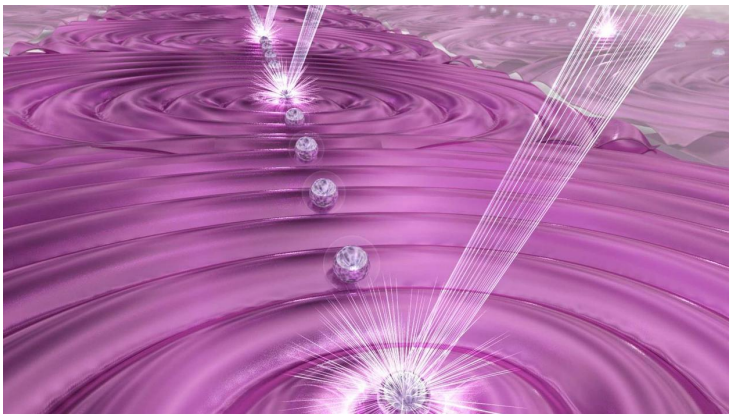

Figure 2: An artist impression of the Maryland ion particles.

## 1.5 Are Quantum Computers Better?

It is widely assumed that a quantum computer is the next great leap in computing technology, however by our current knowledge this is by no means a certainty. There are some very well known benefits to quantum computing, such as significantly faster factorisation (finding two unknown prime numbers that when multiplied together produce a third number that is known) than a normal computer. In 1994 the mathematician Peter Shor demonstrated Shor's algorithm, showing that a quantum computer would be able to solve a factorisation problem orders of magnitude faster[12]. Most online security uses public key encryption based on the fact that a normal computer has virtually zero chance of finding prime factors in a short time frame. Shor showed that quantum computers have the potential to render all online security essentially redundant[13].

Unfortunately there are currently a very small number of quantum algorithms known, with Shor's algorithm and a search method algorithm known as Grover's algorithm being the two most well known of a very short list. This limitation, combined with the fact that any problem that a quantum computer can solve, a classical computer can also solve, makes it unclear when quantum computers will become a viable replacement for normal computers. This is especially true when considering that classical computers will continue to advance according to Moores law[2] for years to come.

# 2 Background Science

## 2.1 Qubits

To begin to understand Quantum Computing it is important to understand what a *qubit* is, as this is the building block upon which all else is built. A qubit is analogous to a bit in classical computational science[10], but with a key set of differences. A classical bit can be either 0 or 1, whereas in true quantum mechanical fashion, a qubit collapses to either its $|0\rangle$ or $|1\rangle$ state when it is 'observed'[6].

A qubit is a two-state quantum system, existing as a 2-component complex vector $|\psi\rangle \in \mathbb{C}^2$. Traditionally this system's eigenstates are denoted as being $|0\rangle$ and $|1\rangle$, and the qubit is a linear superposition of the product of each of these states with the relevant *amplitude*, denoted by $\alpha$ and $\beta$ in equation 1.

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{1}$$

In the realm of quantum computation, a qubit is a wavefunction. To ensure normalisation, a requirement for any quantum mechanical wavefunction, $|\alpha|^2 + |\beta|^2 = 1$ must always be true. Like the eigenstates, the amplitudes $\alpha$ and $\beta$ also exist in the complex domain, $\in \mathbb{C}$.

As a reminder of the physical meaning of these quantities: the probability of measuring the qubit's $|0\rangle$ eigenstate is given by finding $|\alpha|^2$. Below is a visualisation of a qubit which highlights the relation of the qubit to its eigenstates. From Figure 3 , $\alpha$ is given as $\cos\frac{\theta}{2}$ and $\beta$ as $e^{i\phi}\sin\frac{\theta}{2}$.

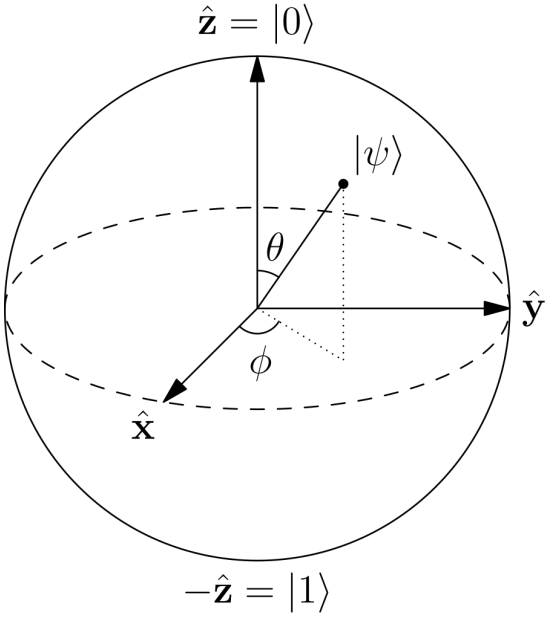


Figure 3: A popular way of visualising a qubit, a Bloch Sphere.

## 2.2 Quantum Registers

Qubits are stored in a quantum register, which is very different from classical data storage. While classical data can be stored in arrays and as a string of bits ('0101001'), the goal of a quantum register is to simultaneously store all the possible values which are spanned by the qubits it contains.

A quantum register stores the tensor product of the individual qubits' vector spaces, and is a vector itself. For example, 3 qubits would be combined as such: $\mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \mathbb{C}^2 = \mathbb{C}^8$, and the resultant quantum register would be an 8-component complex vector. Analogously to the possible combinations for binary representations of integers, the number of dimensions a quantum register containing *n* qubits is required to have is given as $2^n$.

A simple example is useful to show exactly what a quantum register containing 2 simple qubits would look like.

$$|a\rangle = |\psi_0\rangle \otimes |\psi_1\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{2}(|00\rangle - |01\rangle + |11\rangle - |11\rangle) \quad (2)$$

Of course in practice the quantum registers which will be used will not usually be this simple, it is helpful as a way to picture exactly what is occurring.

It is also worth recapping the basic properties of the tensor product to fully understand the function of quantum registers.

- Distributive: It follows the basic rules of multiplication when multiplying brackets.

$$\begin{aligned} \alpha\,|\psi_0\rangle \otimes \beta\,|\psi_1\rangle &= \alpha\beta\,|\psi_0\rangle \otimes |\psi_1\rangle \\ |\psi_0\rangle \otimes (|\psi_1\rangle + |\psi_2\rangle) &= |\psi_0\rangle \otimes |\psi_1\rangle + |\psi_0\rangle \otimes |\psi_2\rangle \end{aligned} \quad (3)$$

- Associative: Positioning of brackets is not important. $|a\rangle\,|b\rangle\,|c\rangle = (|a\rangle\,|b\rangle)\,|c\rangle$

## 2.3 Grover's Algorithm

Grover's Algorithm is an iterative quantum algorithm that is often described as the quantum equivalent of a classical method to search an unsorted database. However, this is a somewhat misleading analogy: it is better described as an algorithm to find the single, unique input for a function, such that the function returns a Boolean true value[14]. Such functions are referred to as *oracle* functions. For the case of Grover's algorithm, these functions are chosen to return -1 for a single, unique input value, and to return 1 otherwise.

The algorithm offers a significant increase in efficiency when compared to equivalent algorithms designed to run on classical computers, being capable of finding the correct input value with a high degree of certainty in less than $\mathcal{O}(\sqrt{N})$ iterations, compared to the $\mathcal{O}(N)$ iterations necessary for the classical equivalent, where $N$ is the number of possible input values for the oracle[15].

The algorithm is as follows[15]:

1. All basis states of the quantum register are combined in a normalised super position, such that, upon measuring, the probabilities of any given basis state being observed are equal.

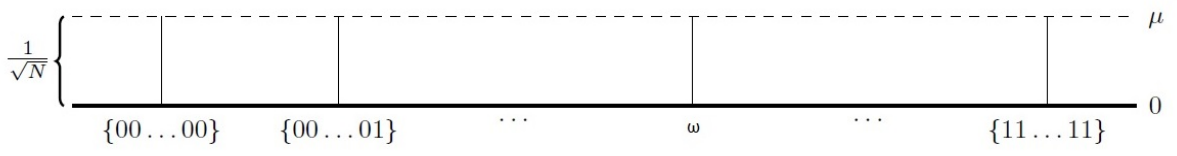


Figure 4: Diagram showing a quantum register in a normalised superposition after the 1st step of Grover's algorithm. Figured adapted from reference [16].

2. The amplitude of the state being searched for is inverted.

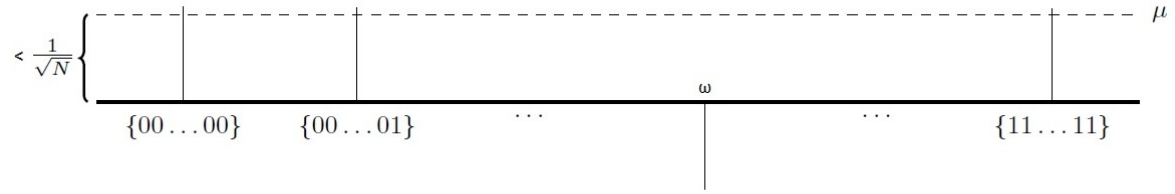


Figure 5: Diagram showing the inversion of the amplitude of the state selected by the oracle, and the effect this has on the mean amplitude of the basis states. Figure modified from reference [16].

3. The mean amplitude of all states is calculated, and the amplitudes of all states are inverted around this mean.

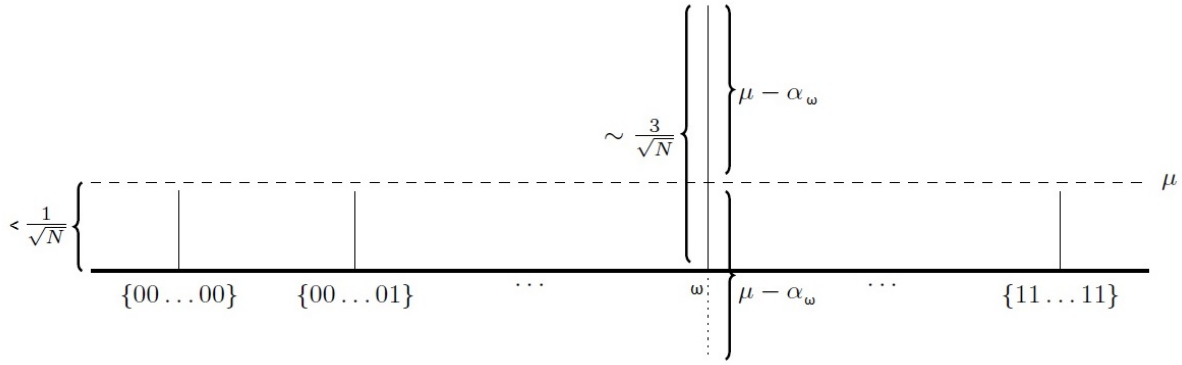


Figure 6: Diagram showing the inversion of amplitudes around the mean, and the resulting difference between the amplitudes of the desired and undesired states, where $\alpha_\omega$ is the amplitude of desired state $|\omega\rangle$. Figure modified from reference [16].

4. Steps 2. and 3. are repeated, increasing the amplitude of the desired state and reducing the amplitudes of other states with each iteration. After a number of iterations, the amplitude of the desired state will be significantly larger than that of the other states, and a measurement of the quantum register will have a very high probability of providing the desired state.

This is more concisely described using Dirac's Bra-ket notation:

$$|\psi\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle \,, \tag{4}$$

where $|x\rangle$ is any given basis state of the quantum register, with each one representing a possible input for the oracle function, and $|\psi\rangle$ is the normalised superposition of basis states, representing all possible inputs for the oracle.

A *quantum oracle operator* $\hat{O}$ is then determined from the oracle function, such that $\hat{O}$ is a $\sqrt{N} \times \sqrt{N}$ square matrix in which all elements on the leading diagonal are 1 other than the element corresponding to the basis state $|\omega\rangle$, representing the desired input value $\omega$, which is -1. All elements off the leading diagonal are zero. This operator is then applied to $|\psi\rangle$, inverting the amplitude of $|\omega\rangle$ but leaving the amplitude of all other basis states unchanged:

$$\hat{O}\left|\psi\right\rangle \equiv \left|\psi_{-}\right\rangle = \frac{1}{\sqrt{N}}\sum_{x=0}^{N-1}(-1)^{\delta_{x\omega}}\left|x\right\rangle, \tag{5}$$

where $\delta_{x\omega}$ is the Kronecker delta piecewise function, defined as:

$$\delta_{x\omega} = \begin{cases} 1, & \text{if } x = \omega, \\ 0, & \text{if } x \neq \omega. \end{cases} \tag{6}$$

The process of inversion around the mean is then performed by applying the *Grover operator* $\hat{G}$ (defined below) to $\left|\psi_{-}\right\rangle$:

$$\hat{G} = 2\left|\psi\right\rangle\left\langle\psi\right| - I, \tag{7}$$

$$\hat{G}\left|\psi_{-}\right\rangle = 2\left|\psi\right\rangle\left\langle\psi\right|\left|\psi_{-}\right\rangle - \left|\psi_{-}\right\rangle. \tag{8}$$

After repeating the application of $\hat{G}\hat{O}$ to $\left|\psi\right\rangle$ $\mathcal{O}(\sqrt{N})$ times, the amplitude of basis state $\left|\omega\right\rangle$ will be significantly greater than that of any other basis state. Measurement of the state of the quantum register at this point will have a very high probability of observing state $\left|\omega\right\rangle$.

However, further repetition of the algorithm will begin to decrease the amplitude of $\left|\omega\right\rangle$, as inverting the amplitude of $\left|\omega\right\rangle$ when it is significantly larger that the amplitudes of other states may result in a negative mean, in turn resulting in a reduction of the amplitude of $\left|\omega\right\rangle$ upon the application of the Grover operator[16]. Hence, it is important to be able to calculate the optimum number of iterations to perform in order to have a high precision in measurement at minimal computational expense. An approximation of the optimal number of iterations can be found using[5]:

$$i \approx \pi\sqrt{\frac{N}{4}}, \tag{9}$$

where $i$ is the number of iterations of the algorithm.

The algorithm is arguably most notable for offering a significant advantage over classical computation when attempting to solve certain NP-complete problems. The NP-complete problems are a set of problems characterised by being extremely difficult to compute an answer, but almost trivial to verify whether or not an answer is correct.

In practice, the algorithm is implemented using a series of *quantum gates* operating on the qubits in a register.

## 2.4 Shor's Algorithm

Shor's algorithm is a quantum algorithm designed to find the prime factors of a given number. This can simplified to the problem of finding a prime factor $d$ for an odd *composite number* $N$, as any even number has a trivial solution of 2. A composite number is one which can be expressed as the product of two numbers, other than one and itself, i.e., a number that is not prime. Using classical computing, the determination of factors of a *sub-prime* (an integer that is the product of two prime numbers) is extremely computationally expensive, with the best known algorithms performing only slightly better than $\mathcal{O}((1+\epsilon)^b)$ for all positive $\epsilon$, where $b$ is

the number of bits in the binary representation of $N$.[17] However, Shor's Algorithm makes use of quantum superposition to solve equivalent problems in polynomial time, vastly outperforming its classical counterparts.

To solve the factorisation problem, it must first be simplified to an order finding problem. Given an integer $M$, it's prime factors can be determined by first finding an integer $r$ such that:

$$r^2 = 1 \ (mod \ M), \ r \neq \pm 1. \tag{10}$$

If such a value of $r$ can be found, then a factor of $M$ can be determined. Since $r^2 - 1 \equiv (r+1)(r-1) = 0 \ (mod \ M)$, $(r+1)$ and $(r-1)$ must be non zero, and factors of a multiple of $M$. Computing the greatest common divisor of $(r \pm 1)$ and $M$ will result in a factor of $M$. This factor can then be tested to determine if it is prime, and if not, can be factored by the method laid out above. The greatest common divisor of two numbers can be easily determined using Euclid's algorithm[18]:

1. Express the larger of the two numbers as the product of the smaller number and an integer, plus a remainder. Consider the example of solving $gcd(27, 33)$:

$$33 = 1 \times 27 + 6 \tag{11}$$

2. Express the smaller of the two original numbers as the product of the first remainder and an integer, plus a remainder.

$$27 = 4 \times 6 + 3 \tag{12}$$

3. Continue this pattern, expressing the factor of the product in the previous step as a multiple of the remainder, e.g.,

$$6 = 3 \times 2 + 0 \tag{13}$$

4. When the remainder reduces to zero, the solution to the greatest common divisor problem is the remainder from the previous step, i.e., $gcd(27, 33) = 3$.

For example, consider the case where $M = 21$:

$$1 \ (mod \ 21) = 22 = 43 = 64... \tag{14}$$

Since 64 is a square number, a valid solution for $r$ is 8. Taking the greatest common divisor of $(r \pm 1)$, the following result is found:

$$gcd((8+1), 21) = 3, \tag{15}$$

$$gcd((8-1), 21) = 7. \tag{16}$$

Since both 3 and 7 are prime numbers, there is no need to iterate the process. The prime factors of 21 have been found to be 3 and 7.

However, determining a non-trivial value for r is more complex. This is done by selecting an integer $A$ at random that is coprime with $M$, such that $1 < A < M$, and finding an integer $s$ such that $A^s = 1 \ (mod \ M)$. As a consequence of number theory, it can be shown that the probability of picking a value of $A$ meeting the latter two conditions from a set of possible coprime values of A is at least $\frac{1}{2}$.[19] If a value of $A$ is selected such that the following conditions are satisfied:

1. $s$ is even
2. $\frac{s}{2} \neq -1 \ (mod \ M)$

then $r$ can be found as:

$$r \equiv A^{\frac{s}{2}} = 1 \ (mod \ M) \tag{17}$$

If $\frac{s}{2} = -1 \ (mod \ M)$, then the factors obtained by find the greatest common divisors will be the trivial factors 1 and $M$.

Suitable values of $A$ can be found by randomly selecting an integer, and classically checking if $gcd(A, M) = 1$. If, after many attempts, a value for $A$ satisfying conditions 1. and 2. cannot be found, then there is a high probability that $M$ is a power of an odd prime number, in which case it can be factorised by searching for $k$ such that $\sqrt[k]{M}$ is a prime number.

The genius of Shor's algorithm is to make use of quantum superposition to solve the order finding problem in polynomial time. This done by determining the period of $s$, i.e. the interval in $s$ over which $A^s \ (mod \ M)$ repeats itself, using a *Quantum Fourier Transform*. The algorithm is as follows[shorsAlgorithmOriginal, 19]:

1. Much like Grover's algorithm, the first step is to initialise the register into a superposition of all basis states:
$$|\psi\rangle = \frac{1}{\sqrt{N}} = \sum_{x=0}^{N-1} |x\rangle \tag{18}$$
2. Define a function $f(x) = A^x \ (Mod \ M)$, and implement this as an oracle function $O_f$. Applying this oracle to $|\psi\rangle$ gives:
$$O_f \, |\psi\rangle = \frac{1}{\sqrt{N}} = \sum_{x=0}^{N-1} |x\rangle \, |f(x)\rangle \tag{19}$$
3. Make a measurement of the value of $|f(x)\rangle$, collapsing the superposition into a value $f(x_0)$ and resulting in the following state:
$$\sqrt{\frac{r}{N}} \sum_{i=0}^{\frac{N}{r}-1} |ir + x_0\rangle \, |f(x_0)\rangle \tag{20}$$
4. Perform a Quantum Fourier Transform on the previous state, resulting in the following state for the register:
$$\frac{1}{\sqrt{r}} \sum_{i=0}^{r-1} |i\frac{N}{r}\rangle \, \phi_i \tag{21}$$
where $\phi_i$ is a phase for each term in the sum resulting from performing a Fourier transform on the shift $x_0$. The state of the register can now be measured to determine a value for a multiple of $\frac{N}{r}$. By repeating the previous steps in the algorithm, several multiples can be obtained and their GCD determined, allowing a value for $r$ to be determined. $M$ can then be factorised by using the same steps outlined previously.

The speed at which Shor's algorithm can factor large subprimes has major implications for the field of cryptography. Many commonly used encryption systems, such as RSA encryption, make use of the fact that it is easy to compute a subprime given too large prime numbers, but

difficult to find two prime factors given a large subprime. RSA does this by encoding messages using a public key, which is a subprime may of two secret large primes, such that the messages can only be decoded using the subprime's factors[20]. If these factors can be determined quickly from the public key, the encryption method is useless. As quantum computing technology improves, cryptographic methods must adapt to cope with a quantum revolution.

## 2.5 Quantum Gates

Much like classical gates, in Quantum Computing there exists a basic set of quantum circuits which form the basis of more complex circuits. Examples of gates used in classical computing include AND, OR, and NOT gates which all act on either one or two classical bits. However, quantum gates have a key set of differences.

Firstly, they have to be reversible. Immediately it is apparent that this requirement has a large effect as it takes away the possibility of quantum analogues to the AND or OR gates as, by definition, the bits given to these gates are irredeemable. Another consequences is the fact the number of qubits which are outputted by the gate have to match the number which are given or inputted.

Secondly, they are typically represented by unitary matrices, whose sizes are given as $2^n \times 2^n$ where n is the number of qubits on which the matrices act, or a square matrix where a column (or row) is the length of the number of dimensions which would be present in a quantum register containing the qubits.

Qubits can also be represented as column vectors which is more appropriate in the discussion of gates, as it is in keeping with the unitary matrix multiplication representation of gates.

$$\nu_{00}\left|00\right\rangle + \nu_{01}\left|01\right\rangle + \nu_{10}\left|10\right\rangle + \nu_{11}\left|11\right\rangle \rightarrow \begin{bmatrix} \nu_{00} \\ \nu_{01} \\ \nu_{10} \\ \nu_{11} \end{bmatrix} \tag{22}$$

### 2.5.1 Hadamard (H) gate

The Hadamard (H) gate is a simple gate which acts on a single qubit. In matrix form it is written as such. Below is shown how the Hadamard gate acts on an eigenstate, denoted as $|x\rangle$.

$$\mathbf{H} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad \mathbf{H}\left|x\right\rangle = \frac{1}{\sqrt{2}}((-1)^x\left|x\right\rangle + \left|1-x\right\rangle) \tag{23}$$

This gate has a wide range of specific uses and is used in a wide variety of quantum algorithms including Grover's Algorithm and Shor's Algorithm. It is used in Grover's Algorithm when the quantum register is initialised and multiple times per iteration, namely after the oracle operator has been applied, and then again after the Grover operator has been applied[5].

A more complete description of the computational steps themselves will be given in the Method section, which details the operation of the quantum computing simulator developed as part of this project.

### 2.5.2 Controlled NOT (CNOT) Gate

A Controlled NOT (CNOT) gate is a quantum gate which is an integral part in the development of a quantum computer. The corresponding classical gate is a reversible XOR gate. It is re-

versible so that the reversibility condition of all quantum gates is satisfied. In matrix form the gate appears as this.

$$\mathbf{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{24}$$

In some cases the gates operation appears to be very similar to its classical analogue but it is used to quantum mechanically entangle two qubits. It flips the target qubit if the control qubit is $|1\rangle$ and does nothing if the control qubit is $|0\rangle$. In order to implement Grover's Algorithm in a physical quantum computer, this kind of gate is required. The Hadamard and Controlled NOT gates together are powerful and a long with a phase shift gate, become a universal set of gates[21] which enable all quantum algorithms to be performed[4–6].

# 3 Program Overview

A simulation of Grover's algorithm was made using Python 2.7. This simulation gave the user the option to simulate the algorithm operating on a register with between two and twenty qubits, and included support for the generation of plots to visualise the performance of the algorithm. Upon running the simulation of Grover's algorithm, the first prompt presented asks the user to enter the number of tests desired. This allows the user to select the number of tests that they would like to perform using the code. A simple prompt that was included for an improved user interface with the code, also preventing a user from having to run the code multiple times. Once the number of tests is selected the user is presented with a list of the different qubits it is possible to run tests for, ranging from 1 to 20. The user is then asked to select a qubit value for each test. If any of the qubit values entered are over 10 then a warning prompt is sent to the terminal informing the user that tests over 10 qubits are very computationally demanding, and in light of this information the user is asked whether they wish to continue with qubit values over 10. Should the user select 'Yes' the code continues as normal, however should the user select no they are presented with a second list of qubits ranging from 1 to 10 and asked once more to select qubit values, from the list, for each of the tests.

After the tests are selected the code prompts a series of preliminary questions, the first of which asks if the user wants to create a file for the data. This gives an option of creating a data file which contains within it the information for each test regarding; number of qubits, number of states, elapsed time, state chosen and the final state measured by the algorithm. The user is asked if they want the code to display figures. This question was included as the code creates a figure plot for each cycle, which can become inconvenient when running a qubit value that uses double and treble figure amount of cycles. Allowing the option for the code to run without displaying figures also allows the user to record an elapsed time that is not disturbed by figure plotting. The following prompt asks whether or not to save the figure, a simple question that allows the figures to be saved in the figures folder, which is especially useful when it is not desired that the code shows the figures. The user is given an option to choose the desired state for each of the tests, and after this the option to choose the number of cycles. This allows the code to either run unaltered, or to run for a specific state over a specific number of cycles, allowing for further usability.

Following the performance of the tests that have been selected the user is asked if they wish to run a graphing module. Selecting yes shows a list of all the output files in the data files folder allowing the user to select which data file will be plotted. After selecting an output file the user

then selects to plot a time performance diagram of the information in the data file. This returns a plot of the number of qubits a test was performed for against the amount of time it took to compute this test in seconds. After running through this final prompt the programme terminates.

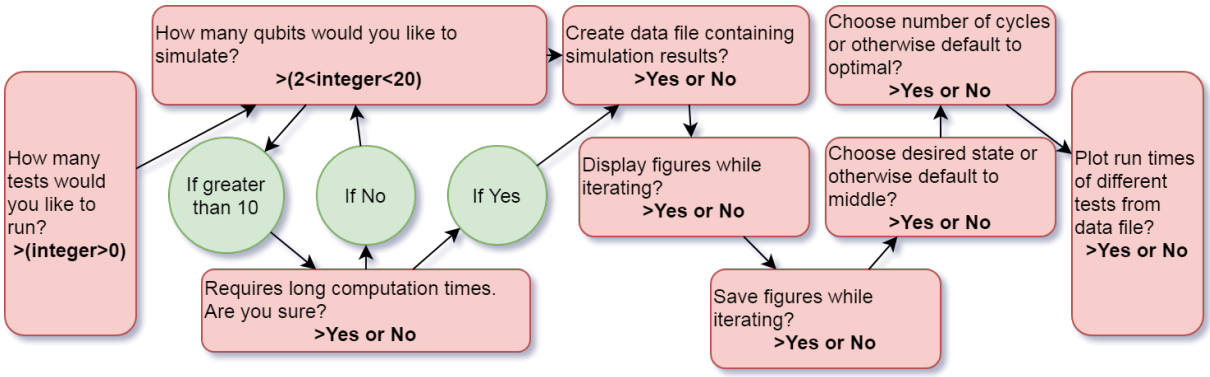


Figure 7: Flowchart of the Program's User Interface.

# 4 Program Design Method

The program was designed with accurate Grover's Algorithm functionality, and user customisability at the forefront. In order to have the large number of simulations options available to the user, and the ability to graph results, many additional lines of codes were necessary which were not directly related to the quantum simulation itself. To prevent the code pertaining to the simulation itself from becoming too obfuscated, the code was partitioned into 4 main modules, and a script at the centre of it all from which the program is run. The contents of each module will be discussed and a visual representation of how the code ties together will also be provided.

## 4.1 Imports Module

The imports module is a succinct and elegant way of keeping track of all the external packages which each module requires. It negates the need of importing every package in the preamble of every module, and instead only the importing of the 'Imports' module is required. This mean it is not needed to keep track of which modules use which packages, and is helpful for organisational purposes.

Packages the code requires include NumPy, which is used to create the matrices representing qubits, quantum registers, and quantum gates. One of the benefits of NumPy is that much of its backend code is written in either C/C++ or Fortran[22], meaning that using inbuilt functions to perform operations such as matrix applications is significantly faster than methods written in Python itself. Matplotlib and SciPy are also used for plotting purposes[23].

## 4.2 Auxiliary Classes Module

The Auxiliary Classes Modules contains questions used as part of the user interface and initialises the possible qubits which can be simulated. This is where the warning about the demands of simulating large numbers of qubits is located. The data file object to which simulation results are written class is also located here.

## 4.3 Preliminary Questions Module

This module is used to abstract a lot of code away from the Main function; specifically to contain all the 'Yes' and 'No' questions, as well as a function to sanitise the inputs and account for special cases. All these questions are detailed above in the Program Overview section, eg. the option to display figures or not. Answers are handled by being appended to a list where each index corresponds to specific question. This list is the main output of the module and is what is returned to the Main function of the program.

The function which sanitises the 'yes' or 'no' inputs has a special case for when the question of creating a data file is being asked. If this question is being asked, it is in this function where a Data File object is initialised to store and save the data generated during the simulation. This is performed here as it is a simple line to include, and serves to not distract from the flow of the Main function.

## 4.4 Quantum Classes Module

The Quantum Classes Module is where the majority of the quantum simulation code is located. The parent class is a Quantum Register object. The constructor creates three fields when this object is initialised. The first field is a list of the qubits which will be simulated as specified by the user in the Main function. From this list it is possible to calculate the number of states, $2^q$ which is the next field. The third field is a complex NumPy array which contains all the possible qubit states, the entries in a quantum register as described in the introductory sections.

The class also contains a method to pass the register through a Hadamard (H) gate, which is technically a Hadamard transform on the whole register. It works backwards through the quantum register and uses the NumPy Kronecker product of the gate on each entry and itself to create a Hadamard transformation matrix of the correct size. Once this has been worked few, this gate, or transform, multiplies the qubit states. This has the effect of performing the Hadamard (H) gates.

Additionally, there is a simple function to calculate the probabilities, which are simply the state amplitudes squared. This is also where the figure plotting and saving code is contained.

Grover's Algorithm is then a subclass which extends the Quantum Register class and implements specific methods used for Grover's algorithm. The methods include the oracle operator, which inverts the correct state in the register and the conditional phase shift operator which inverts every element of the register.

## 4.5 The Main Function

The Main function or QuantumRun.py is where the program is ran. First all the questions are asked sequentially, with the majority of their processing taking place in the backend modules. Once the simulation parameters are defined by the user, the key function in this script, the quantumLoop where the iterations for Grover's algorithm can be performed.
The computational steps which go into each cycle of Grover's Algorithm is to first measure the probabilities and the oracle operator, a Hadamard transform, the conditional phase shift, and then another Hadamard transform. This matches the steps described in above in the theoretical description of the algorithm given above, and successfully selects the desired state.

## 4.6 Graphing Module

The graphing module is an optional module which asks to be run after the simulation has finished if a data output file is to be plotted. Currently the plotting option is the run time of the simulation for different numbers of qubits. It is designed with the idea of making it easily expandable should it be decided that further graphing modes are needed at a later date. This is a sophisticated way of plotting data as it ties directly to the main program and circumvents the issue of running separate scripts for plotting and specifying the correct files.

## 4.7 Design Diagram

A design diagram is included to give a visual intuition of how each of these modules connects. A connecting line represents a direct dependency between one part and another. Rounded yellow rectangles are used to represent modules, blue rectangles to represent classes, purple ellipses for functions and finally the rounded green rectangle at the centre is the script from which the program is run.

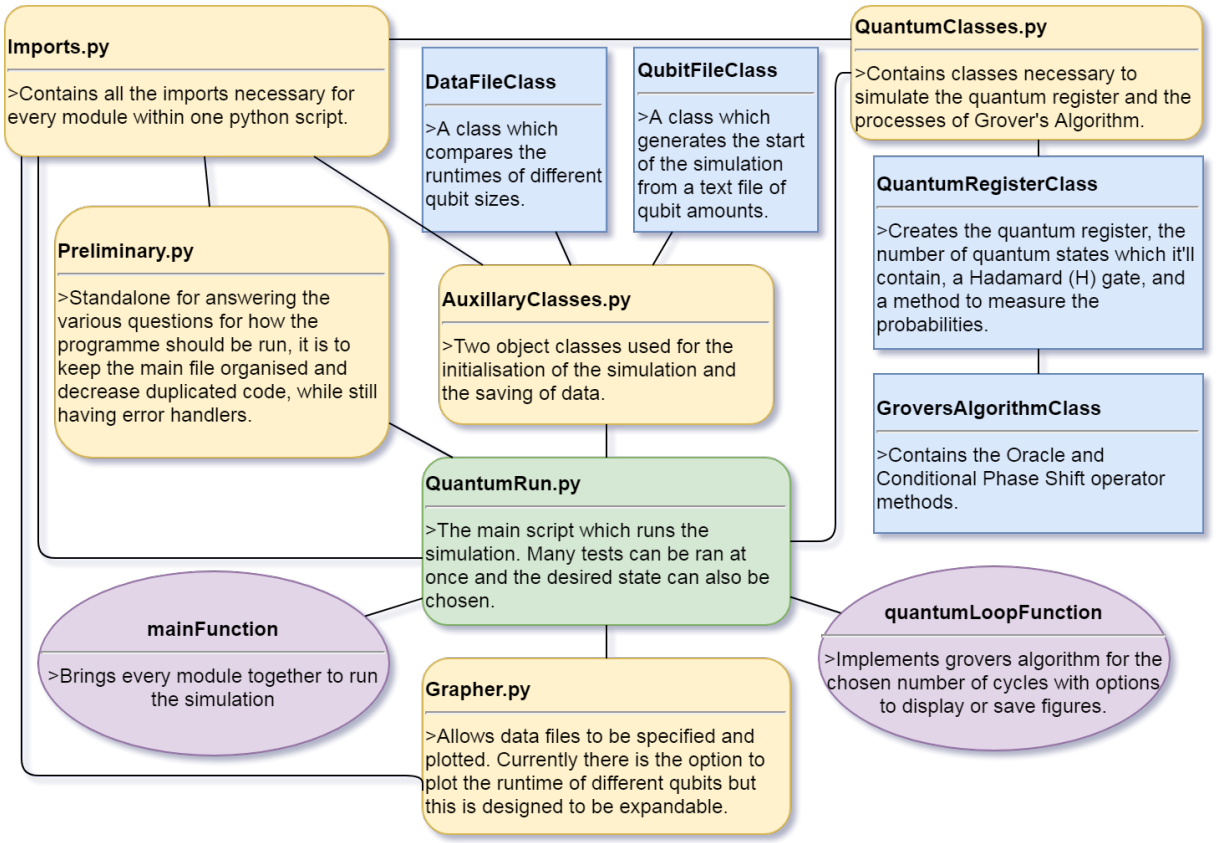


Figure 8: A diagram of how the modules connect and the program is run.

# 5 Results

## 5.1 Five Qubits: Optimal Cycle Length

The graphs here plot, for each cycle, the probability for each state of the superposition of states collapsing into that state upon observation. With the state numbers along the x axis and probability along the y axis, Figure 9 shows a graph for every cycle performed with the 5 qubit

system. In this system, $(2^5 =)32$ possible states are created. The chosen state for this simulation was the case where no desired state is specified by the user, which causes the program to default to half of maximum state number, in this case, 17.

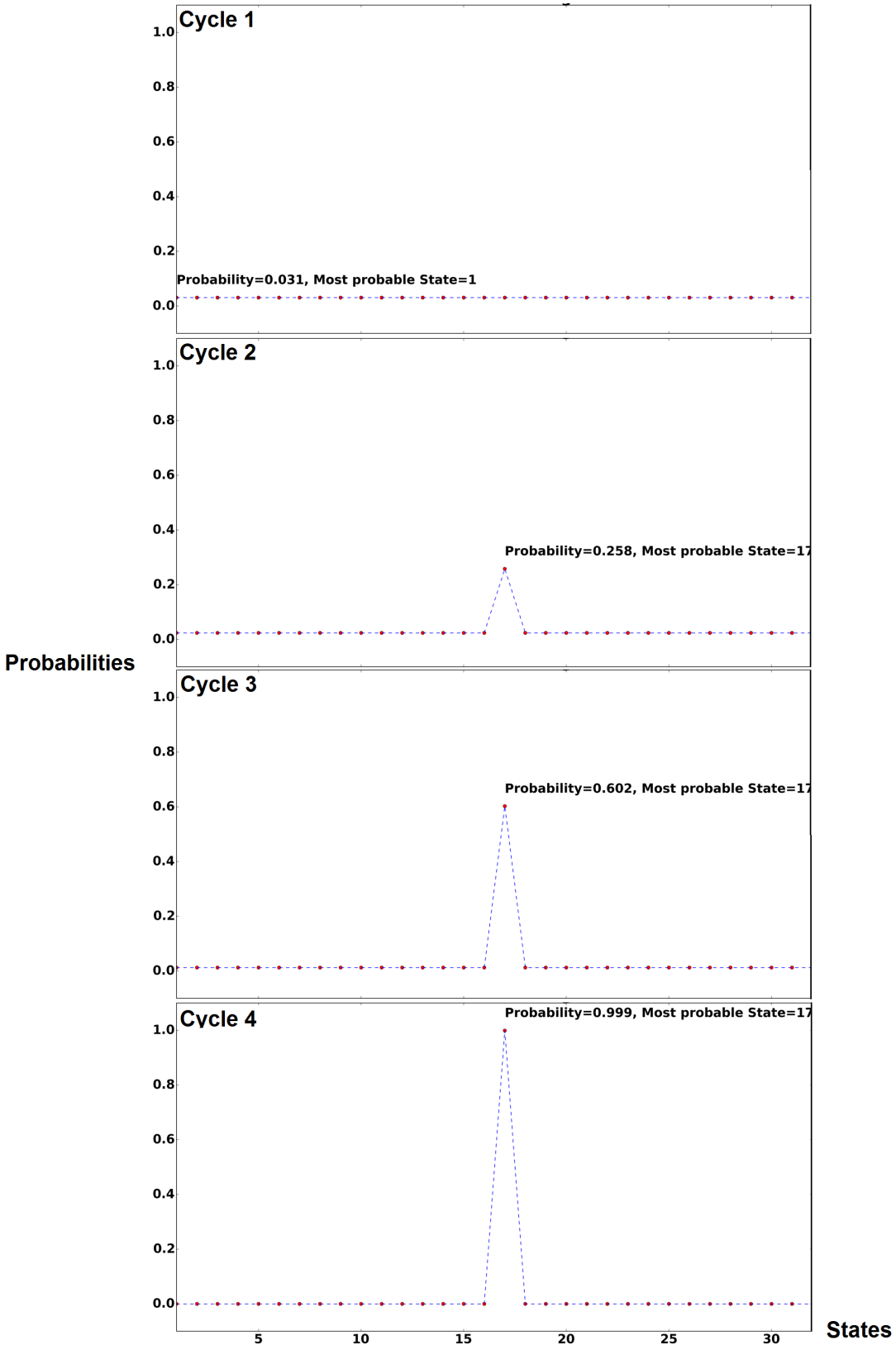


Figure 9: The optimal cycle length of Grover's Algorithm for 5 qubits.

As can be seen in the Figure 9 the program has successfully selected the desired state and the probability of observation of that state increases with each cycle, before finally achieving a

value of a probability of 99.9%. This is as expected for a working version of Grover's algorithm. Due to the probabilistic nature of Grover's algorithm this value is never exactly one but should be incredibly close to one, and this is successfully implemented in the code.

One of the key features of Grovers algorithm as highlighted in section 2.3 is that as the probability of the desired state increases, the probabilities of all the other states decreases. This can be seen in the graphs, as in the first cycle all the states have the same probability value of 0.031. In the second cycle it can be seen that the other states have a probability value that is still non zero, but has decreased due to the increase in probability of the desired state. This again indicates a successful implementation of Grovers algorithm.

## 5.2 Five Qubits: Excessive Number of Iterations

The three-dimensional plots in Figure 10 show how a 5 quibt system progresses when run for more than optimal number of cycles. For each plot, the x axis shows the probability of measurement of any given state, the y axis shows the cycle number, and the z axis shows the state number. As described in Section 2.3 the optimal number of cycles can be approximated using equation 9.

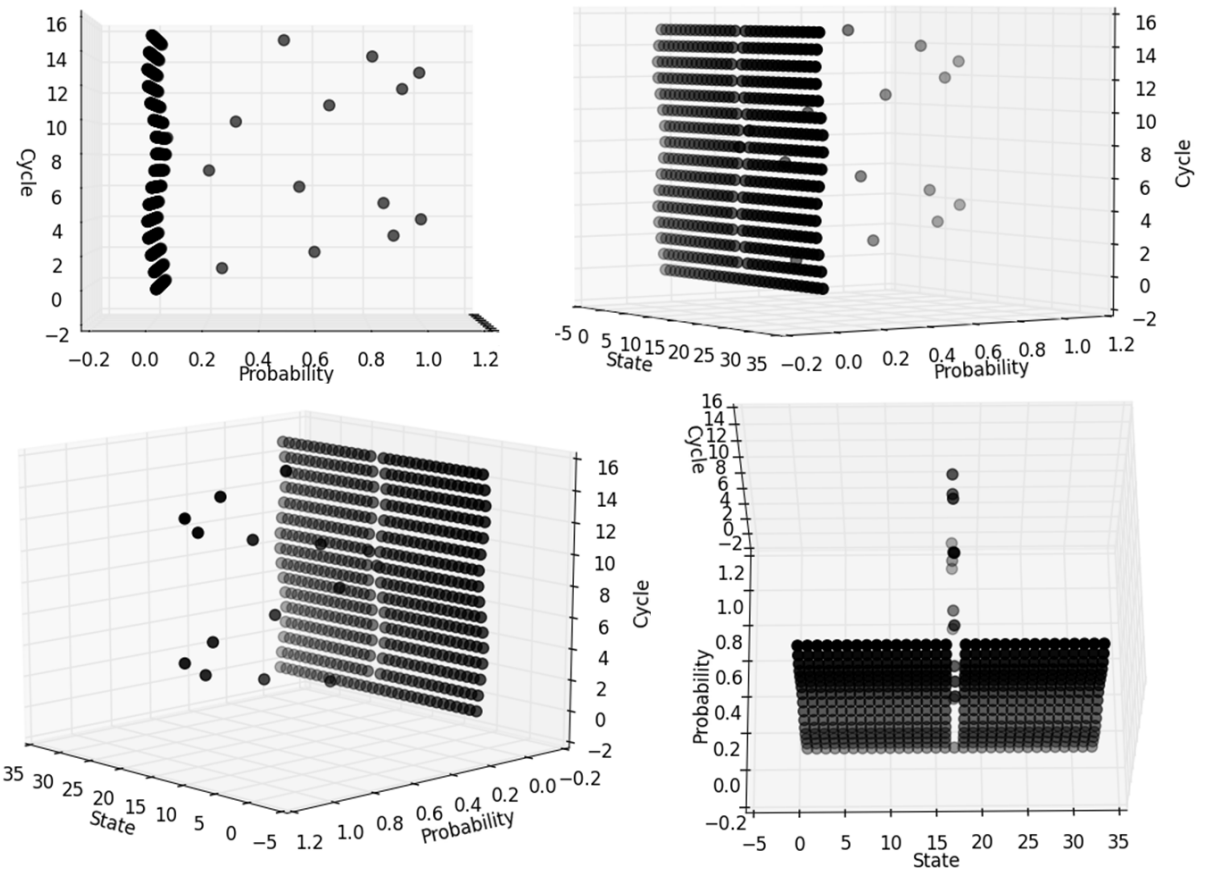


Figure 10: 3D representation of the iterations of Grover's Algorithm. 5

As can be seen from the plots, the probability of observing the desired state increases as the cycle number approaches the optimum number, before decreasing again. This is a result of the amplitude of the desired state becoming so large, that when inverted, the mean amplitude across all states becomes negative, resulting in the reduction of the amplitude of desired state after performing inversion around the mean. When run for yet more cycles the amplitude of the desired state follows a wave-like pattern, decreasing after the optimum number of cycles

until the register returns to a state very similar to the initial superposition, before once again increasing towards a maximum. This behaviour is to be expected with Grover's algorithm, and once again is evidence for a successful implementation.

## 5.3 Fourteen Qubits: At the Limits of Processing Power

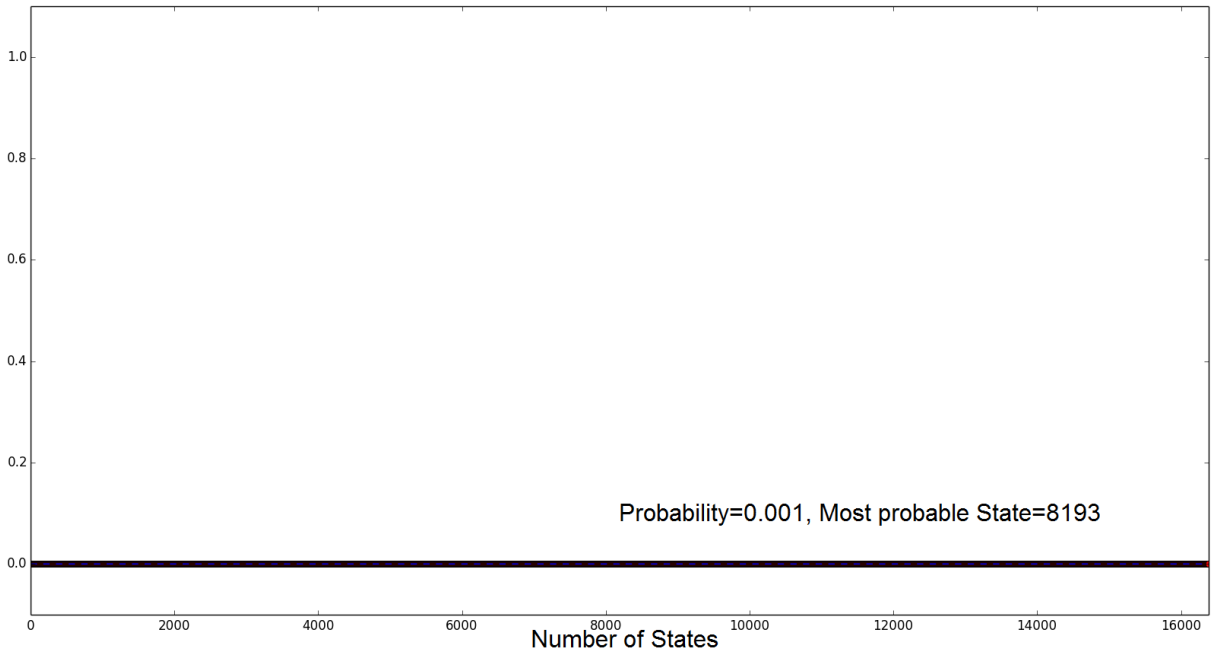


Figure 11: 14 Qubit system after the 2nd cycle.

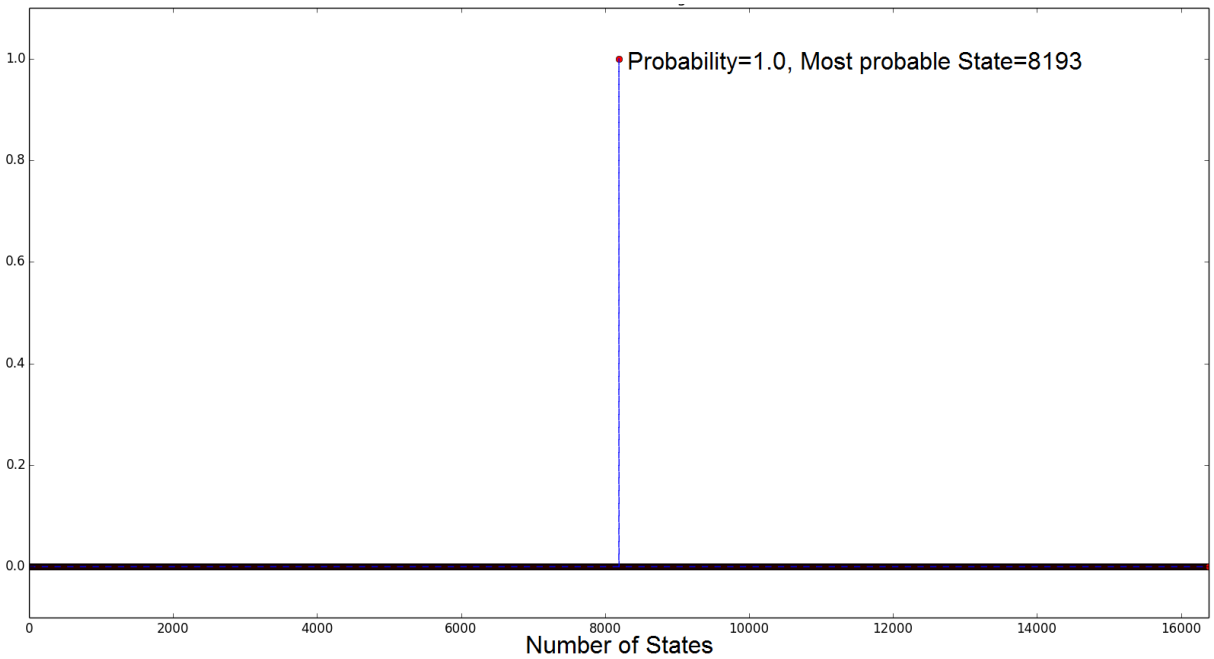


Figure 12: 14 Qubit system after the 101st cycle.

The axes are the same as previously described, with the x axis showing the state's value and the y axis showing the probability of measurement. In the second cycle, the probability of observing the desired state can be seen to have increase by an extremely small value, compared

to the increase displayed by the 5 qubit system. However, after many cycles, the desired state achieves a probability of 1.0 after the 101$^{st}$ cycle (the actual value is in fact slightly less than one due to the probabilistic nature of Grover's algorithm, but rounds to 1 when rounded to one decimal place). This shows that the simulation scales to large numbers of qubits correctly, which the number of cycles required matching the estimation provided by equation 9.

## 5.4 The Scaling of Computation Times

Figure 13 shows a plot of the computational time in seconds against the number of qubits being tested. As can be seen for qubit values up to 10 the computation times are small. For qubit values greater than 10, the execution times start to jump at an increasing fast rate, this exponential growth in the time taken is to be expected. With the number of states increasing to the power of n, each cycle becomes more and more computationally demanding, and a greater number of cycles is required to find the desired state with optimum accuracy. Included on the graph are the qubits ranging from 2 to 14. While the code does have the potential to run 15 qubits and higher, it becomes so computationally demanding that most consumer computers are unable to complete the algorithm. 15 qubits was successfully performed on a computer that managed to complete the cycles in 8666 seconds, however for the purpose of demonstrating results this value has not been included. This exponential growth in execution time is a consequence of modelling the algorithm using classical methods. By making use of quantum parallelism, an ideal quantum computer would see no increase in execution time for a single cycle with increasing numbers of qubits, with the only increase in execution time of the algorithm being as a result of having to perform more cycles for larger registers.

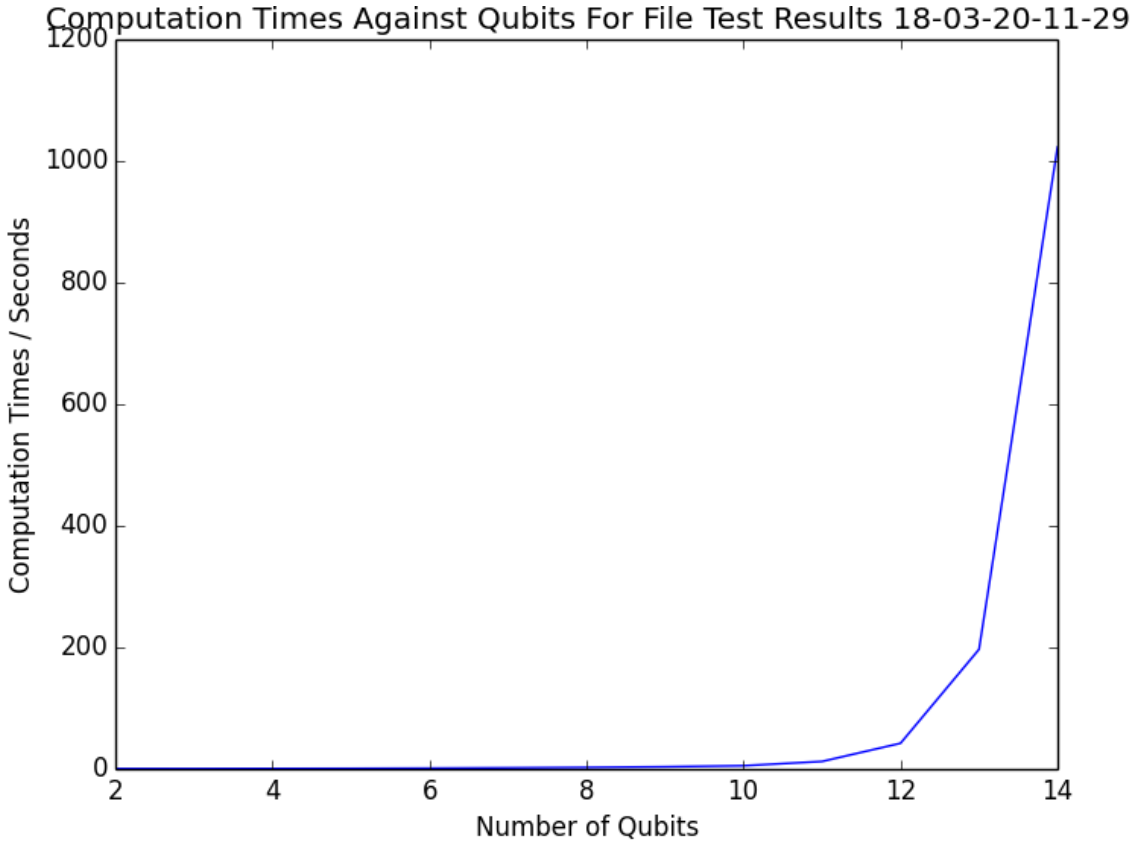


Figure 13: Plot of the computational time versus the qubit number

# 6 Real Quantum Computers

Whilst Grover's algorithm can be modelled on a classical computer relatively simply, implementing a working quantum computer to run the algorithm is an far more difficult task. In 2017, a three qubit grover's search was implemented using a programmable quantum computer, using trapped atomic ions[24]. This was the first example of Grover's algorithm running with more than 2 qubits on a scalable quantum computer.

The computer was implemented by trapping laser cooled positive Ytterbium ions in a magnetic field to act as qubits, which were manupulated by pusled lasers operating as quantum gates.

The quantum algorithm was compared to an equivalent classical algorithm by running both algorithms for a single iteration, and comparing the probability of determining a correct solution for each. A classical three bit search for a single desired value can only ever attain a probability of 25% after one iteration, as the best a classical algorithm can achieve is to check one entry to see if it is the desired value, and if not, guess another value at random. However, Grover's algorithm was found to identify the desired state with a probability of 78.125% after a single iteration[24]. Similarly, when searching for one of two desired states, the classical algorithm achieved an accuracy of just 46.4%, whilst Grover's algorithm found one of the desired states after one iteration with a probability of 100%.

Whilst a Grover's search of a three qubit system offers little practical application, the improvements over classical algorithms are clearly demonstrated, and as such systems are scaled to larger and larger numbers of qubits, these improvements will become significant in the solution of real-world problems.

# 7 The Future of Quantum Computation

The prospect of real-life quantum computation has been recognised as being highly advantageous to many research fields, and perhaps to society as a whole, but as of yet the construction of a large scale quantum computer has not been made feasible[25]. This has been an ongoing area of research since as early as the 1980s with numerous designs for physical machines.

This section will look at one of the most recent, prominent designs for a quantum computer based around microwave trapped ions[25]. This was published only last year and has already been gaining traction in the field with some hailing it as the "first viable blueprint of a quantum computer"[26].

The basic premise of this blueprint which separates it from previous designs is that it is fully scalable, and consists of many individual components which can be easily connected, thus solving many of the problems previously faced by existing quantum computer designs. These individual components are quite complex in design but are simple in principle.

The components, or modules, are micro-fabricated chips which contain atomic ions trapped by microwaves in many junctions called X-Junctions, and are the unit cells of the quantum computer.

These X-Junctions are comprised of three zones, one for loading and trapping an ion, another zone where microwave-based gate operations can be performed on a set of ions and a readout zone where an ion's state can be read using a global laser. Ions can be transported from the

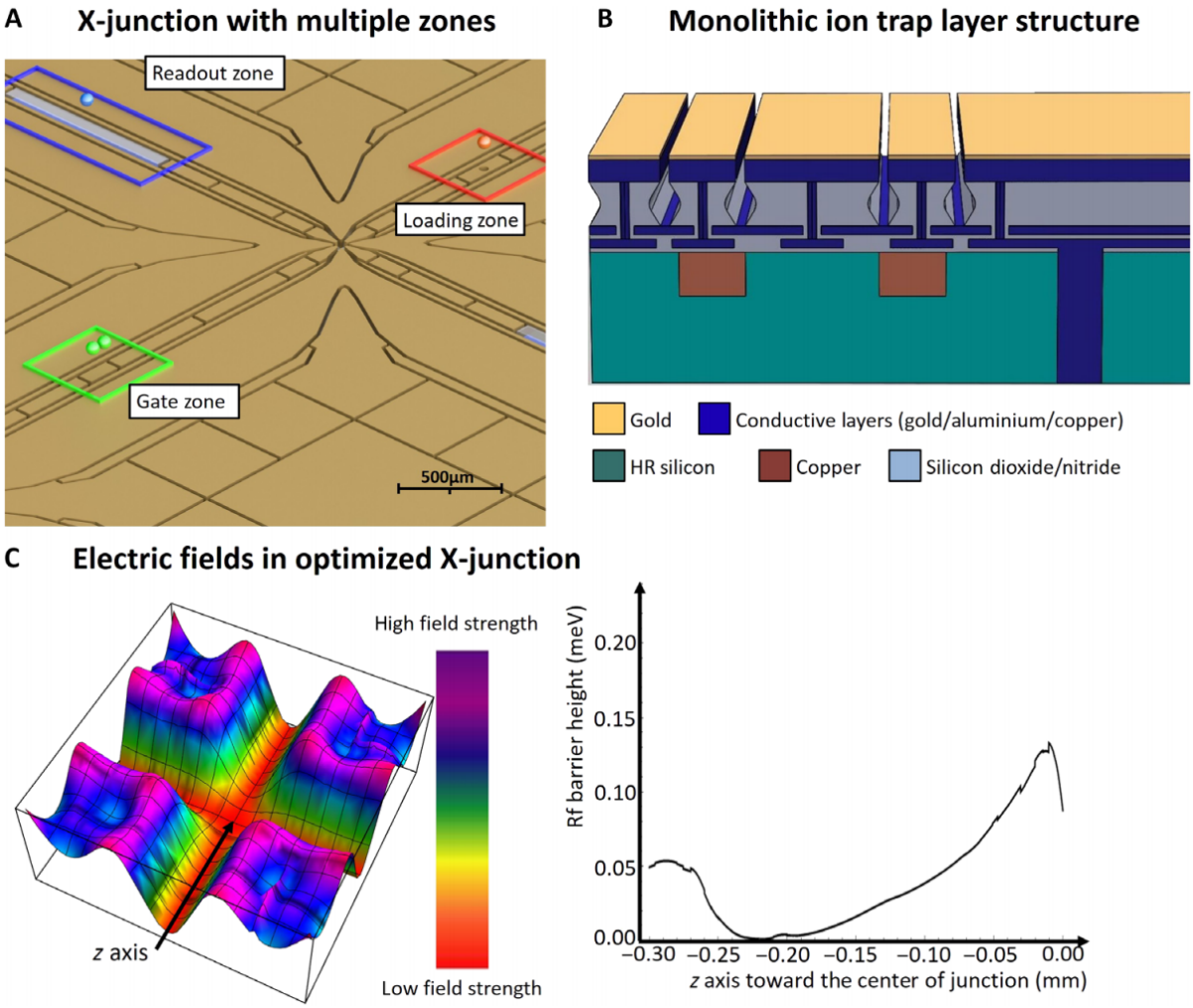


Figure 14: A schematic of the X-Junction its underlying electric fields[25].

loading zone to the gate zone using ion shuttling operations, and then magnetic fields can be used to address them individually, as well as entangle them with the aid of electric fields. State detection is also made possible by then shuttling an ion to the readout zone. The main benefit of only using a global laser for the process of measuring states is that the laser requirements for measurement are far less stringent than those imposed on lasers used to realise quantum gates.

Atomic ions have long been recognised as suitable candidates for quantum computers[27] as they are a quantum system which is often not only stable but also well defined. They also allow logical gate operations to be performed through utilisation of the strong Coulombic interactions between ions to couple different qubits. Some argue that trapped ions are the best candidate for the physical realisation of qubits on a large scale[28].

$^{43}Ca^{+}$ is an atomic ion which exists as two hyperfine states, and can be trapped using radiofrequency electric fields. With a modestly sized applied magnetic field, the energy gap between the two states is sufficiently large, increasing the discernibility of the two states. This 'easy discernibility' is also known as being high-fidelity, an important feature as this overcomes problems associated with working in noisy environments where similar qubit states could otherwise be confused.

The X-Junctions proposed in this blueprint have been specifically designed inline with current capabilities of microfabrication techniques and to tackle problems previously faced in the design

of quantum computers. Part of their appeal is the how they can be aligned so that huge numbers of ions/qubits can interact computationally.

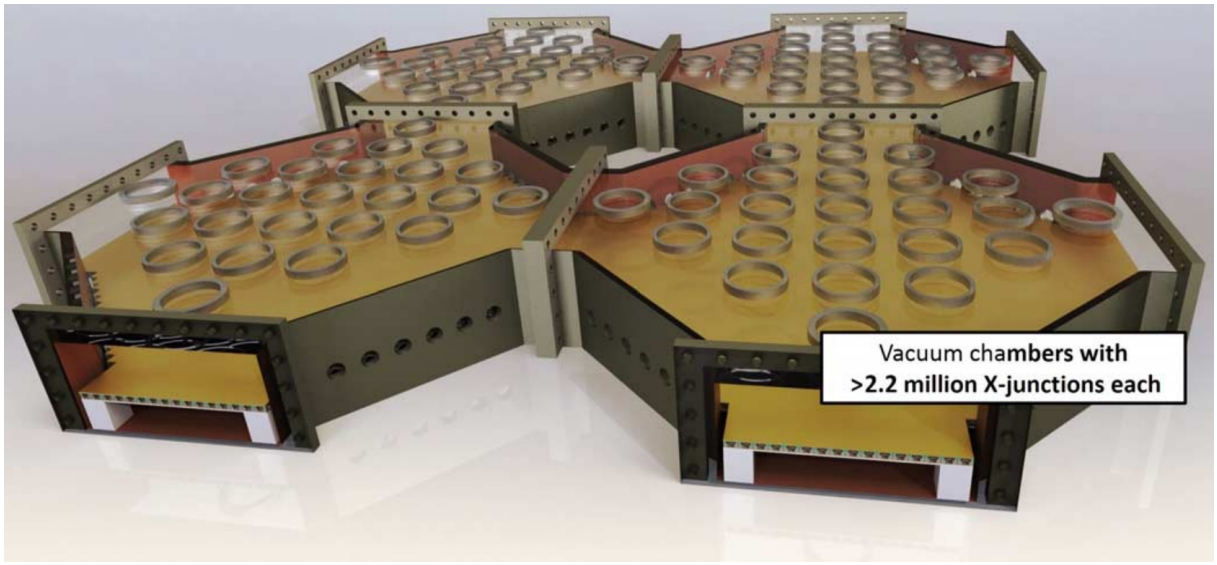


Figure 15: Combining many vacuum chambers containing many X-Junctions

The team specifically proposed a design where 2.2 million X-Junctions could exist within vacuum chambers, as well as describing mechanisms by which these chambers could connect, thus scaling the possible system sizes to be even larger. Finally, they performed vigorous theoretical testing on a computational model of the blueprint and found it could withstand a realistic amount of misalignment of internal modules[25].

# 8 Project Execution Strategy

In order to carry out a project of this magnitude it was required that the bulk of the workload was distributed evenly among the team. A natural course of action was to divide the work into two sub teams where one worked on report writing and the other worked on the writing of the program itself.

A. Chattarji was chosen to head the team which worked on the program itself due to his Computer Science credentials and F. Elie provided him with assistance. E. Morrison, A. Reid, and J. Smith worked on the report writing and worked on a very even basis, dividing sections into whichever interested each member most.

Meetings for both subteams occurred usually on a weekly basis, occurring more frequently as the deadline drew closer and additional meetings were taken place regularly for the entire group, such that both teams could liaise. The preliminary entire group meetings were used for developing the understanding conceptually of the quantum computing techniques which were going to be employed as this was necessary for both the report and the program writing teams.

As the program was nearing completion, entire team meetings were used to fully explain the functionality and design of the program so that the report writing sub team could accurately describe the program itself. Additionally a Facebook group, an online shared LaTeX document and a shared Dropbox was created and used to keep everyone up to date with what was decided at each meeting and what the focus of each week was going to be.

This execution strategy proved to be highly successful as the project brief was met and the program provided a large amount of customisability. The project could have been improved if the team had focused on understanding Shor's algorithm conceptually sooner, which would have given the program design sub team a longer time to extend the program to include this algorithm as well.

# 9 Conclusion

The two main aims of this paper were firstly to introduce the concept of quantum computation; and to present a simulation of a quantum computer running Grover's algorithm and the results that this simulator obtained. Quantum computers were introduced by first discussing classical computer construction and data representation using transistors. The motivation for building quantum computers was introduced, presenting Moore's law of transistor density and the fact that transistor density must approach a limit when transistors reach a quantum scale. The introduction also briefly discussed the mechanisms by which quantum computers would use qubits and, as well as highlighting some of the advantages quantum computation offers over its classical counterpart, the paper presented an example of a 53 qubit quantum simulator that has been built and explained the physics behind how it works.

This rather surface level introduction was then followed with a more in depth discussion of the back ground science. This started with a discussion of qubits in which Dirac notation was used to introduce the mathematics behind the concept. Following this, quantum registers were introduced, discussed and an example of the mathematics used to combine qubits into a register was presented. Using the concepts of qubits and quantum registers Grover's algorithm was explained to the reader, and formulated mathematically. One key function of this section was to present the expected results of Grover's algorithm so that the results of our own quantum simulator could be compared to the theoretical expectations. Following this was a discussion of a more complex quantum algorithm, Shor's algorithm. Shor's algorithm is a method of finding two unknown prime factors of a third known number, a method that classical computers fall very short in performing[29]. This section of the paper also included an introduction to quantum gates, discussing how they differ from classic computational gates as well as presenting an example known as the Hadamard gate. The Hadamard gate as shown in equation 23 has a wide range of uses including Grover's algorithm.

The second key aim of the paper was to present a simulation of Grover's algorithm created in Python 2.7. Firstly, to do this, a short programme overview of the code's user interface is presented along with a flow chart explaining the options a user has to choose from. Following this there is an in depth discussion of the program design methods, starts with a discussion of the various modules that were set up to create the simulation. These modules include: an import module, that imports all the required code packages; an auxiliary class module that sets up the user interface questions; as well as a preliminary questions module that uses the user's answers to choose what implementations the code will perform. The quantum classes module is then explained: this class contains the key implementations of Grover's algorithm including the Hadamard gate. As well as explaining the main function of the code and the graphing module the paper also presents a design diagram to show how the different modules and classes work together.

Following this the paper presents the results of the Grover's algorithm simulator. The results start by showing all four cycles of a 5 qubit system and discussing different features of the

graphs, in relation to how they show that the code successfully implemented the algorithm. The results then use a 3D plot to visually represent what happens when a system of qubits is run past its optimal cycle number. These plots demonstrate another key feature of Grover's that after the optimal number of iterations has been reached the probability of the desired state reduces again. Also presented in the results are iterations from a 14 qubit system to demonstrate some of the potential the simulator created has. After this there is a plot that shows how long, in seconds, each of the different qubit systems take to reach their optimal iteration. Having such a successful implementation of Grover's algorithm the paper concludes that quantum computers have great potential to be the future of computational technology, and along side this the paper finishes by presenting the potential future of quantum computers.

# 10 References


[1]Jeffrey Carter, *Moore's law figure*, Accessed 7th March 2018, `http://pointsandfigures.com/2015/04/18/moores-law/`.

[2]L. B. Kish, "End of moores law: thermal (noise) death of integration in micro and nano electronics", Physics Letters A **305** (2002).

[3]I. Chuang and M. Nielsen, *Quantum Computation and Quantum Information* (Cambridge University Press, 2010).

[4]N. D. Mermin, *Quantum Computer Science: An Introduction* (Cambridge University Press, 2007).

[5]R. T. Perry, *Quantum Computing from the Ground Up* (World Scientific Publishing Co., 2012).

[6]A. Ekert, P. Hayden, and H. Inamori, "Basic concepts in quantum computation", Coherent Atomic Matter Waves **72** (2001).

[7]R. Landauer, "The physical nature of information", Physics Letters A **217** (1996).

[8]R. P. Feynman, "Simulating physics with computers", International Journal of Theoretical Physics **21** (1982).

[9]D. Deutsch, "Quantum theory, the church-turing principle and the universal quantum computer", Proceedings of the Royal Society of London A **400** (1985).

[10]A. Galindo and M. Martin-Delgado, "Information and computation: classical and quantum aspects", Reviews of Modern Physics **74** (2002).

[11]J. e. a. Zhang, "Observation of a many-body dynamical phase transition with a 53-qubit quantum simulator", Nature **551** (2017).

[12]P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer", SIAM Journal on Computing **26** (1996).

[13]D. Leermakers and B. Skoric, "Optimal attacks on qubit-based quantum key recycling", Quantum Information Processing **17** (2018).

[14]Craig Gidney, *Grover's quantum search algorithm*, Accessed 7th March 2018, `http://twistedoakstudios.com/blog/Post2644_grovers-quantum-search-algorithm`.

[15]L. K. Grover, "A fast quantum mechanical algorithm for database search", STOC '96, 212–219 (1996).

[16]J. Wright and T. Tseng, *Lecture Notes on Quantum Computation: Grover's Algorithm* (Carnegie Mellon University, 2015).

[17]R. Crandall and C. Pomerance, *Prime Numbers: A Computational Perspective* (Springer, 2001).

[18]H. Cohen, *A Course in Computational Algebraic Number Theory* (Springer, 2013).

[19]U. Vazirani, *Lecture Notes on Shor's Factoring Algorithm* (University of California Berkeley, 2004).

[20]R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", Commun. ACM **21**, 120–126 (1978).

[21]R. Li, M. Hoover, and F. Gaitan, "High fidelity universal set of quantum gates using non-adiabatic rapid passage", Quantum Information Processing **9** (2009).

[22]SciPy Developers, *Numpy and scipy documentation*, Accessed 8th March 2018, `https://docs.scipy.org/doc/`.

[23]J. Hunter, D. Dale, E. Firing, M. Droettboom, and T Eam, *Matplotlib documentation*, Accessed 8th March 2018, `https://matplotlib.org/contents.html`.

[24]K. A.L.N.M.L.S. D. C. Figgatt D. Maslov and C. Monroe, "Complete 3-qubit grover search on a programmable quantum computer", Nature Communications **8** (2017).

[25]B. Lekitsch, S. Weidt, A. G. Fowler, K. Mølmer, S. J. Devitt, C. Wunderlich, and W. K. Hensinger, "Blueprint for a microwave trapped ion quantum computer", Science Advances **3** (2017).

[26]The University of Sussex, *First ever blueprint unveiled to construct a large scale quantum computer*, Accessed 10th March 2018, `http://www.sussex.ac.uk/broadcast/read/38900`.

[27]D. Kielpinski, C. Monroe, and D. J. Wineland, "Architecture for a large-scale ion-trap quantum computer", Nature **417** (2002).

[28]J. Kim, "Viewpoint: trapped ions make impeccable qubits", Physics **7** (2014).

[29]R. Jozsa and A. Ekert, "Quantum computation and shor's factoring algorithm", Reviews of Modern Physics **68** (1996).