Shor's Algorithm

Talha Ashraf

Abstract

In this paper, we will cover the fundamentals of Quantum computing and use them to analyze Shor's factoring algorithm. This will involve showing that factorization can be reduced to the problem of order finding, which leads to a Quantum algorithm that can prime factorize an integer in polynomial time complexity.

1 Introduction

Peter Shor's 1974 paper sent shockwaves throughout the field of cryptography as it threatened to allow Quantum computers to break our most relied on encryption methods. RSA and ECC encryption schemes are designed on the bases that computers cannot efficiently factorize large numbers or compute discrete logarithms respectively. Shor's paper showed algorithms for doing precisely these two tasks efficiently on a Quantum computer. For this paper, I will focus on describing his factoring algorithm.

Shor's algorithm is not just fast, its exponentially faster than classical computers. The most efficient classical algorithms for factoring numbers take exponential run time while Shor accomplishes this task in mere polynomial time. The mechanic of Quantum computers that gives Shor's algorithm such a leg up is the principle of superposition. This principle states that Quantum systems do not just have to exist in one of their base states, they can exist in an intermediate state from where they have a probability of collapsing into any one of their base states based on probability.

This principle is of so much utility because we can encode information into these superpositions. If you want to compute every possible value of a function for example, then in a single cycle of a Quantum computer, you can create a superposition which contains every possible value of the function. This sounds too good to be true and gives the impression that Quantum computers can magically do thousands of calculations at once. Indeed, there is a catch: each value of the function is very difficult to retrieve from the superposition.

But this issue does not completely kill hope for utilising the vast potential of superposition as a means of doing many simultaneous calculations. There are clever ways of constructing Quantum algorithms that can extract usable information from superpositions in a Quantum computer.

In this paper, we will start off with covering the fundamental concepts of Quantum computing. To this end, we will formalise our concept of superposition, define the qubit -or "Quantum bit"- and learn about Quantum gates for manipulating qubits. We will apply this knowledge to constructing

Deutsch's algorithm which finds properties of a function by constructing a superposition of all its values.

Then, we will proceed to developing the core tools used in Shor's algorithm. These tools are the Quantum Fourier Transform and the Quantum Phase Estimation procedure. Both these algorithms are central to computing orders in the order finding problem. This is a problem which is closely related and can be used to achieve the task of factoring numbers.

Finally, we will see how to bring everything together, and write an algorithm on a Quantum computer that wil, with high probability, factor large numbers in polynomial time through order finding. You can find Peter Shor's original paper and his exposition on the algorithm in [Sho97].

2 Superposition: Defining the Qubit

The principle of Superposition is perhaps the most important concept in Quantum computing because it is the key difference between information on a classical computer versus that on a Quantum computer.

A qubit is the fundamental object for handling information on a Quantum computer. The best way understand a qubit is to refer back to classical computers, which store information as a series of 1s and 0s called bits. These bits can be used to represent general information, for example we can ascribe each possible permutation of bits to an integer using the following mapping: $x_n x_{n-1} \dots x_2 x_1 \to x_n \cdot 2^{n-1} + x_{n-1} \cdot 2^{n-2} + \dots x_2 \cdot 2^1 + x_1 \cdot 2^0$ where $x_j \in \{0, 1\}$. Since this is a bijective map, all integers can be represented with a sufficiently large number of bits. Importantly, classical bits are binary, each bit can be either 0 or 1. If you take a hundred bits and inspect each of them they will all be observed to represent the same digit of either 0 or 1.

A qubit can similarly exist in definite states. Let us work generally and say we have a qubit which can be either in the state called $|a\rangle$ or the state called $|b\rangle$. We will discuss what these strange $|\rangle$ brackets mean later. If you measure a qubit in state $|a\rangle$ with respect to the states $|a\rangle$ and $|b\rangle$, you will definitely observe it to be in state $|a\rangle$. It is the same for a qubit in state $|b\rangle$.

There is one striking, key difference between qubits and classical bits. Qubits are not restricted to these basic definitive states. They may also exist in what is called a superposition of these deinite states, which can be written as

$$|\psi\rangle = \alpha |a\rangle + \beta |b\rangle . \tag{1}$$

where α and β are any two complex numbers for which $|\alpha|^2 + |\beta|^2 = 1$ where $|\alpha|^2 = \alpha\alpha^*$. This state $|\psi\rangle$ is neither $|a\rangle$, nor $|b\rangle$, and it may help you to think of it as a distinct state of its own. It is, however, strongly related to the definite states $|a\rangle$ and $|b\rangle$. To be specific, whereas when you measure $|a\rangle$ or $|b\rangle$ you observe $|a\rangle$ or $|b\rangle$ with 100% certainty, when you measure the superposition $|\psi\rangle$ with respect to the states $|a\rangle$ or $|b\rangle$, you will have some probability of observing the state $|a\rangle$ and some probability of observing the state $|b\rangle$. After you measure a superposition, it 'collapses' by which I mean, it stops being a superposition and takes on the deinite state that you observed. The pprobability that the superposition collapses to $|a\rangle$ is $|\alpha|^2$ and the probability it collapses to $|b\rangle$ is $|\beta|^2$.

THEOREM 1. ***Collapse of a Superposition:*** *If a qubit in the superposition $|\psi\rangle = \alpha\,|a\rangle + \beta\,|b\rangle$ is measured with respect to the states $|a\rangle$ and $|b\rangle$, it collapses into either the state $|\psi_{measured}\rangle = |a\rangle$ with probability $|\alpha|^2$ or the state $|\psi_{measured}\rangle = |b\rangle$ with probability $|\beta|^2$ and stays in this collapsed state for the future. In order to conserve probability, we must normalise the complex amplitudes so that $|\alpha|^2 + |\beta|^2 = 1$.*

For our purposes here, we do not need to discuss the Physics behind what constitues a measurement, we simply think of it as the the operation that collapses a qubit. Measurement depends on what basis we are measuring with respect to. Measuring against the states $|a\rangle$ and $|b\rangle$ returns either $|a\rangle$ or $|b\rangle$. Measuring against a different set of states, say $|c\rangle$ and $|d\rangle$, will return either $|c\rangle$ or $|d\rangle$

The set of basic states that we can use to represent a qubit or a superposition of a qubit and which we ultimately measure against is called a basis.The most fundamental basis for a singe qubit is the set of states $|0\rangle$ and $|1\rangle$. This is called the computational basis. You can write any other valid basis in terms of the computational basis. It plays a very similar role to the unit vectors $\hat{x}$ and $\hat{y}$ in representing points in a 2d plane.

Since the output from a quantum program must be a definite string of binary bits, a superposition is not a valid output itself. In order to get a final output that we can actually make sense of, we must always collapse any superposition by measuring it. Traditionally, final measurements are done against the computational basis so that we either get an output of $|0\rangle$ or $|1\rangle$ for each qubit. These final output states are then treated as string of binary $0s$ and $1s$ and used to store information just like classical bits do on a nornal computer.

3 Change of Basis

Let us discuss the $|\rangle$ brackets. This is called ket notation. A ket is used to signal that the variable in the ket is a vector in a complex vector space. In order to explicitly express $|\psi\rangle$ as a vector, notice from equation (1) that in order to fully specify the state $|\psi\rangle$ all we need are the complex constants α and β. Let's work in the computational basis now and set $|a\rangle = |0\rangle$ and $|b\rangle = |1\rangle$. Knowledge of α and β tells us everything about the probability of $|\psi\rangle$ collapsing into each state. In other words, all we need to know the exact state $|\psi\rangle$ is α and β and the fact we are in the computational basis. In light of this, we can represent $|\psi\rangle$ as a column vector of these two constants:

$$|\psi\rangle = \begin{bmatrix}\alpha\\ \beta\end{bmatrix} \tag{2}$$

Then, we select vectors to represent $|0\rangle$ and $|1\rangle$ so that equations (1) and (2) are consistent with each other:

$$|\psi\rangle = \alpha\begin{bmatrix}1\\0\end{bmatrix} + \beta\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}\alpha\\ \beta\end{bmatrix}$$

$$|0\rangle = \begin{bmatrix}1\\0\end{bmatrix} \qquad |1\rangle = \begin{bmatrix}0\\1\end{bmatrix}$$

Another basis that comes up very often is the Fourier basis, which we will discuss constantly throughout this paper. For a single qubit system, the Fourier basis is as follows:

$$|+\rangle \equiv \frac{|0\rangle + |1\rangle}{\sqrt{2}} \qquad |-\rangle \equiv \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

It is easy to see how we can convert from the computational to the Fourier basis:

$$|\psi\rangle = \alpha\,|0\rangle + |1\rangle = \alpha\frac{|+\rangle + |-\rangle}{\sqrt{2}} + \beta\frac{|+\rangle - |-\rangle}{\sqrt{2}} = \frac{\alpha+\beta}{\sqrt{2}}\,|+\rangle + \frac{\alpha-\beta}{\sqrt{2}}\,|-\rangle$$

It is also important to see the result of measuring a qubit in the states $|+\rangle$ and $|-\rangle$ against the computational basis. For these states, $\alpha = \frac{1}{\sqrt{2}}$ and $\beta = \pm\frac{1}{\sqrt{2}}$ so that $|\alpha|^2 = |\beta|^2 = \frac{1}{2}$. So the Fourier basis for a single qubit is like a coin flip; it represents a 50/50 chance of measuring $|0\rangle$ or $|1\rangle$.

4 Multiple Qubits

So far we have talked about only a single qubit. We can generalise to a larger system composed of multiple qubits. When we increase the number of qubits in our system, we are also increasing the size of the vector space that our state vectors reside in. This also means we need a new computational basis with a larger number of basis vectors to represent superpositions of our qubit system. The operation that is used when scaling to a larger vector space is the tensor product. We will not go in depth with tensor products, and only introduce what is necessary.

In a classical computer, with more bits come more permutations of bits. With 3 bits, for instance, you have the options $000, 001, 010, 011, 100, 101, 110, 111$. These permutations guide us on what vectors to choose for our computational basis for a 3 qubit system:

$$\begin{aligned} |\psi\rangle =& \alpha_{000}\,|000\rangle + \alpha_{001}\,|001\rangle + \alpha_{010}\,|010\rangle + \ldots \alpha_{111}\,|111\rangle \\ =& \alpha_{000}\begin{bmatrix}1\\0\\\vdots\\0\end{bmatrix} + \alpha_{001}\begin{bmatrix}0\\1\\\vdots\\0\end{bmatrix} + \ldots \alpha_{111}\begin{bmatrix}0\\0\\\vdots\\1\end{bmatrix} = \begin{bmatrix}\alpha_{000}\\\alpha_{001}\\\vdots\\\alpha_{111}\end{bmatrix} \end{aligned}$$

Really, a multi-qubit system is just several single qubit systems pooled together. In this context, a tensor product is just the operation that is joining these qubits into a larger system which can represent more information. Here's an example of taking a tensor product of two qubits:

$$|0\rangle \otimes |1\rangle = |01\rangle = \begin{bmatrix}1\\0\end{bmatrix} \otimes \begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$$

.

For a system of n qubits, taking a tensor product gives us column vectors with 2^n rows. This means we have 2^n basis vectors, and thus 2^n complex constants in any superposition. This exponential growth of vectors allows us to represent a large amount of information with relatively few qubits. A system of 50 qubits means we have 2^{50} complex constants which is a ginormous number.

Qubits follow similar properties to plane multiplication. Here is an example of how to take the tensor product of two superpositions:

$$\begin{aligned}(\alpha |0\rangle + \beta |1\rangle) \otimes (\gamma |0\rangle + \delta |1\rangle) =& \alpha\gamma |0\rangle \otimes |0\rangle + \alpha\delta |0\rangle \otimes |1\rangle + \beta\gamma |1\rangle \otimes |0\rangle + \beta\delta |1\rangle \otimes |1\rangle \\ =& \alpha\gamma |00\rangle + \alpha\delta |01\rangle + \beta\gamma |10\rangle + \beta\delta |11\rangle\end{aligned}$$

For this paper, I will alternate between three different formats for representing systems of qubits. The formats

$$|x_1 x_2 \dots x_n\rangle = |x_1\rangle |x_2\rangle \dots |x_n\rangle = |x_1\rangle \otimes |x_2\rangle \otimes \dots |x_n\rangle \tag{3}$$

all represent the same thing, but it often gets very messy to write every single $\otimes$ sign explicitly in long equations so I will often drop them in favour of the first two formats.

5 Quantum Gates

Now that we are familiar with the quintisencial behaviour of qubits, we can finally work on devising the building blocks of Quantum circuits and algorithms: Quantum gates.

In a classical computer, there are logic gates, which take a set of bits as an input and output another set of bits. The truth tables of the most common logic gates (the NOT, OR, AND and XOR gates) are as follows:

NOT Gate		OR Gate			AND Gate			XOR Gate		
0	1	0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	1	0	0	1	1
		1	0	1	1	0	0	1	0	1
		1	1	1	1	1	1	1	1	0

Table 1: Classical logic gates.

In a Quantum computer, we also need some sort of gates to manipulate information. It turns out that Quantum gates satisfy some very special properties. Firstly, a quantum gate, U, acting on qubits results in a linear transformation. This means that for any two states $|\psi\rangle$ and $|\phi\rangle$, we have that

$$U(\alpha |\psi\rangle + \beta |\phi\rangle) = \alpha U |\psi\rangle + \beta U |\phi\rangle . \tag{4}$$

This may sound obvious at first sight, but a quick look at classical gates shows that they do not obey this. For example $G_{and}10 = G_{and}01 = 0$ but $G_{and}(10 + 01) = G_{AND}11 = 1$. Here G_{and} is a function implementing the the AND gate.

Similarly to superposition, this linearity is directly inherited from Quantum Mechanics where the central Schrodinger Equation is famously linear, leading to linearity in many Quantum phenomena.

This is great for us because all linear transformations on finite dimensional vectors can be represented as a matrix, which means that we can represent any Quantum gate with an associated matrix. More specificallly, a Quantum gate cannot be just any matrix, they are a special type of matrix known as a unitary matrix. All unitary matrices are invertible, which means any set of quantum gates are also invertible and any operation on a set of qubits can be reversed!

Now, we can finally formulate some Quantum gates. Our first gate will be the Quantum NOT gate, represented by the symbol X. This gate will take a single qubit input and flip its value: $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$. Through the linearity of this gate, we can see the result of having an X gate act on a qubit in superposition:

$$X(\alpha|0\rangle + \beta|1\rangle) = \alpha X|0\rangle + \beta X|1\rangle = \alpha|1\rangle + \beta|0\rangle .$$

Thus in a sense, the Quantum NOT gate is switching the positions of the constants α and β. The metric representation of this gate is as follows:

$$X \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix}$$

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} .$$

Do you remember the Fourier basis I mentioned earlier? Perhaps the most used Quantum gate is the Hadamard gate H, which can be used to put a qubit into the $|+\rangle$ or $|-\rangle$ states. More specifically, $H|0\rangle = |+\rangle = \frac{\alpha|0\rangle+\beta|1\rangle}{\sqrt{2}}$ and $H|1\rangle = |-\rangle = \frac{|0\rangle-|1\rangle}{\sqrt{2}}$.

You should be able to show that the Hadamard gate is represented by the matrix

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} .$$

One last class of Quantum gates I will discuss are controlled gates. Every Quantum gate has a control variant. In a control gate, there is a central unitary operation that we would like to perform on a certain qubit called the target bit. But there is also a second qubit passed to the gate called the control qubit. The role of the control qubit is to determine whether the unitary operation should go through. If the control qubit is in the state $|1\rangle$, the unitary operator U acts on the target bit. However, if the conrtol qubit is in the state $|1\rangle$, the target qubit passes through the gate unchanged.

It is very nice to be able to represent Quantum circuits diagramatically. Figure 1 should give you a flavour for how we draw Quantum gates. Note that for control gates, the control qubit is always marked with that black dot. We will use the notation

$$U[x_{control}]x_{target}$$

to repesent application of controlled U gates.

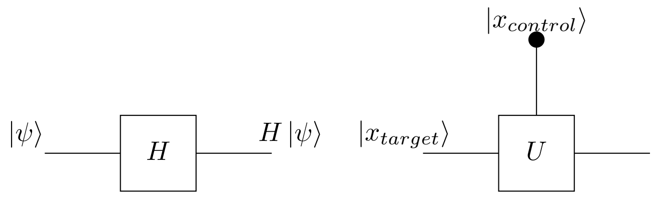

Figure 1: Control gate

6 Quantum Parallelism

The key difference between classical and Quantum computation is the ability of a qubit to be in superposition, and it is thus this principle that we must use to devise efficient Quantum algorithms,

Quantum parallelism is an idea that has garnered a lot of media hype. Most explanations of the phenomenon are somewhat misleading. It seems many people are of the idea that Quantum computers can magically do thousands of computations simultaneously by creating a massive superposition of every permutation of the problem being solved.

There is some truth to this statement. Quantum computers can in fact evaluate a function or problem for many different input values at once via superposition, where a system of qubits is put in a superposition where each state encodes some information about our problem. The immediate issue for us, however, is that even if every possible solution of our problem is correctly represented in some state of the superposition of qubits, that does not mean we have this information immediately at hand.

Remember that the final output we read after measuring a qubit is just the state that the superposition collapses into. Thus a single measurement only nets us information about a single state, not the rest of the superposition at large. Worse yet, which state the superposition will collapse into after measurement is probabilistic. Hence, Quantum computers can't output the results of thousands of computations simultaneously. Even if we save immense computational time by embedding the solutions to a problem into a superposition, we will lose this time when we have to measure the qubits over and over again to get our answers.

Problem 1. *We can see these ideas in action through an example. Suppose we have a function with a one-bit domain and range: $f : \{0, 1\} \to \{0, 1\}$. What want to know the following: Is $f(x)$ constant or balanced? In other words is $f(0) = f(1)$ or is $f(0) \neq f(1)$?*

We do not know what the function is, but we have been provided with a reversible circuit that can compute f. Think of this circuit as a black box of sorts, you enter your input x, and you get out your output $f(x)$, but you don't know the inner workings of this circuit, so the only way to find each output of the function on a classical computer is to run the black box for every x.

2 cycles on a classical computer can give us the values $f(0)$ and $f(1)$ and then we can determine if these values are equal to find the answer to our question.

We can conduct the same 2 cycle operation on the Quantum computer by modeling this classical

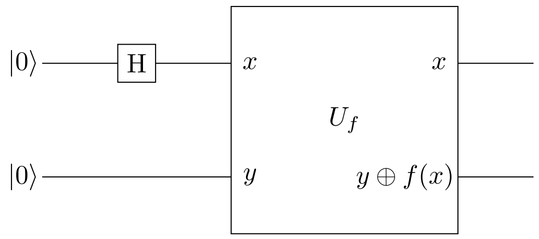

Figure 2: Circuit to create a superposition of states for $f(x)$.

circuit using unitary gates. But there is a more efficient method through Deutsch's algorithm which utilises superposition.

Given the blackbox for finding $f(x)$, we can construct a unitary matrix that takes two inputs and has the effect

$$U_f : |x, y\rangle \to |x, y \oplus f(x)\rangle \tag{5}$$

If $y = 0$, then it is easy to see that $y \oplus f(x) = f(x)$. Hence we set y to $|0\rangle$. In this way we have essentially recovered the classical algorithm where $U_f |x, 0\rangle = |x, 0 \oplus f(x)\rangle = |x, f(x)\rangle$. We can do better though. Consider setting x to $|+\rangle$ by passing it through a Hadamard gate before using it as an input for U_f. Now we have

$$U_f\Big(H|0\rangle\Big)|0\rangle = U_f\Big(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\Big)|0\rangle = \frac{U_f|0\rangle|0\rangle + U_f|1\rangle|0\rangle}{\sqrt{2}}$$

where $x = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$. Hence, our final state becomes

$$\frac{|0, f(0)\rangle + |1, f(1)\rangle}{\sqrt{2}}. \tag{6}$$

If we measiure this superposition, we have a 50/50 chance of measuring either $|0, f(0)\rangle$ or $|1, f(1)\rangle$. Hence in a single cycle of our Quantum circuit, we have managed to create a superposition with every possible value of $f(x)$. In a sense, our qubit system has evaluated $f(x)$ for every value x simultaneously. But as eluded to earlier, this simultaneous evaluation is not saving us computational time because when we collapse the superposition we either measure the state $|0, f(0)\rangle$ or $|1, f(1)\rangle$ each with a $\frac{1}{2}$ chance. This means that in order to know both $f(0)$ and $f(1)$ we would have to conduct this cycle and measure the final superposition at least 2 times.

If our goal was to find the specific values of $f(0)$ and $f(1)$, then Quantum parallelism really is not helping us here. We would be better off modeling the classical algorithm using unitary gates. However, with a few extra steps from here, we can create a Quantum circuit that can find $f(0) \oplus f(1)$ in a single cycle. The trick to do this is to make it so that the quantity $f(0) \oplus f(1)$ is a part of every state of the superposition so that whichever state it collapses onto, we find out the value of $f(0) \oplus f(1)$. If f is constant, then $f(0) \oplus f(1) = 0$ and if it is balanced, then $f(0) \oplus f(1) = 1$.

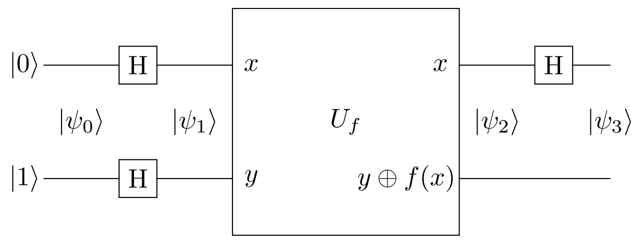

Figure 3: Circuit implementing Deutsch's algorithm.

7 Deutsch's Algorithm

Look at figure 3, which implements Deutsch's algorithm. Let us go through each step along the circuit, described by the states $|\psi_0\rangle$,$|\psi_1\rangle$, $|\psi_2\rangle$ and $|\psi_3\rangle$ where we start out with initial state $|\psi_0\rangle = |0\rangle|1\rangle$. Through the action of the Hadamard gates, we find $|\psi_1\rangle$:

$$\begin{aligned}|\psi_1\rangle = H|0\rangle H|1\rangle = |+\rangle|-\rangle &= \frac{|0\rangle + |1\rangle}{\sqrt{2}}|-\rangle \\ &= \frac{1}{\sqrt{2}}\Big(|0\rangle|-\rangle + |1\rangle|-\rangle\Big).\end{aligned}$$

Now we must apply the unitary matrix U_f to this state. Remember that matrices are linear, so the gate can be ditributed as follows:

$$|\psi_2\rangle = U_f|\psi_1\rangle = \frac{1}{\sqrt{2}}\Big(U_f|0\rangle|-\rangle + U_f|1\rangle|-\rangle\Big) = \frac{1}{2}\Big(U_f|0\rangle|0\rangle - U_f|0\rangle|1\rangle + U_f|1\rangle|0\rangle - U_f|1\rangle|1\rangle\Big)$$

Notice the presence of two terms of the form $U_f(|x\rangle|0\rangle - |x\rangle|1\rangle)$. These two terms can be rewritten in the following form:

$$|x\rangle|0 \oplus f(x)\rangle - |x\rangle|1 \oplus f(x)\rangle = |x\rangle|f(x)\rangle - |x\rangle|\overline{f(x)}\rangle$$

Notice that if $f(x) = 0$, then the expression is $|x\rangle|0\rangle - |x\rangle|1\rangle = |x\rangle|-\rangle$, while if $f(x) = 1$, it is $|x\rangle|1\rangle - |x\rangle|0\rangle = -|x\rangle|-\rangle$. So, the term $|x\rangle|-\rangle$ stays the same while the sign in front of it changes based on the value of $f(x)$. Hence, it is a simple next step to write the expression as

$$U_f(|x\rangle|0\rangle - |x\rangle|1\rangle) = (-1)^{f(x)}|x\rangle|-\rangle\,.$$

With this, we can finally write $|\psi_2\rangle$ in the following simplified way:

$$|\psi_2\rangle = \frac{1}{2}\Big((-1)^{f(0)}|0\rangle|-\rangle + (-1)^{f(1)}|1\rangle|-\rangle\Big)$$

Now, we arrive at two cases which give us two different outcomes. We can either have $f(0) = f(1)$ or $f(0) \neq f(1)$ which correspond to the function being constant or balanced respectively and which lead to different outcomes for $|\psi_2\rangle$:

$$|\psi_2\rangle = \begin{cases} \pm|+\rangle|-\rangle & \text{if } f(0) = f(1) \\ \pm|-\rangle|-\rangle & \text{if } f(0) \neq f(1) \end{cases} \tag{7}$$

Now all we have to do is to apply the Hadamard gate to the first qubit and measure these states. After the Hadamard gate, the first qubit will transform into $|0\rangle$ for the case of $f(0) = f(1)$ or $|1\rangle$ for the case of $f(0) \neq f(1)$.

$$|\psi_3\rangle = \begin{cases} \pm |0\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} & \text{if } f(0) = f(1) \\ \pm |1\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} & \text{if } f(0) \neq f(1) \end{cases} \tag{8}$$

You can equally well write the final state in the more compact form $|\psi_3\rangle = \pm |f(0) \oplus f(1)\rangle |-\rangle$. In either case, the key is that this algorithm allows us to find out whether or not our function f is balanced in a single cycle of our Quantum computer. The first qubit of our 2 qubit system is in the state $|f(0) \oplus f(1)\rangle$ so when we measure the final state $|\psi_3\rangle$, if we find that the first qubit collapses to $|0\rangle$, we know the function was constant while if it collapses to $|1\rangle$, we know that it was balanced. This algorithm has half the runtime of its classical counterpart, which took two cycles to find the answer.

You may find discussion of the previous ideas and the more general Deutsch Jozsa algorithm through their original paper at [DJ92].

8 The Quantum Fourier Transform

So far, we have established many of the foundations of Quantum computing and have familiarised ourselves with the ability of Quantum computers to solve for many different values at once. Now are ready to finally dig into the inner workings of Shor's factoring algorithm. First, however, we must develop the idea of phase estimation, which is central to Shor's algorithm. In turn, to estimate phases, we must also study the Quantum Fourier Transform.

Here is the problem we are trying to solve for this section and the next: Given some unitary matrix U, and its eigenstate $|u\rangle$, what is a good approximation for the phase of its eigenvalue? To illuminate on this question a little, the eigenstate of a matrix is a vector which, when transformed by the matrix, stays the same and gains a scalar factor. More specifically, $U|u\rangle = \lambda |u\rangle = e^{i2\pi\phi} |u\rangle$. The reason that the eigenvalue of our unitary matrix, λ, is $e^{i2\pi\phi}$ is that a unitary matrix preserves the norm of vectors it acts on, meaning the eignenvalue must have magnitude of 1. The most general complex number with a magnitude of 1 is $e^{i\theta}$.

It turns out that there is a a fairly simple Quantum algorithm which uses control gates implementing the operator U to take n qubits, and transform them into the superposition

$$\frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{i\frac{2\pi\phi y}{N}} |y\rangle = \frac{1}{2^{\frac{n}{2}}} \sum_{y=0}^{2^n-1} e^{i\frac{2\pi\phi y}{N}} |y\rangle \tag{9}$$

This expression probably looks very odd to you at the moment, that is because I have not yet defined what many of the variables here mean. I will do so shortly. We will also discuss the Quantum algorithm that implemented this transformation in the following section on Phase Estimation. For now, I want you to try and notice how similar the expression in equation (9) is to the discrete

Fourier transform:

$$\frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{i2\pi\phi y} y \tag{10}$$

There exists an efficient Quantum algorithm (which we will construct in this section) that can take the computational basis vectors and transform them into the the same expression we got in equation (9) in polynomial time. More specifically, our algorithm denoted by U_{QFT} takes a n-qubit vector in the computational basis $|x\rangle = |x_1 x_2 \ldots x_n\rangle$, $x_i = \{0, 1\}$ and transforms it into

$$U_{QFT} |x\rangle = \frac{1}{2^{\frac{n}{2}}} \sum_{y=0}^{2^n-1} e^{i\frac{2\pi xy}{N}} |y\rangle . \tag{11}$$

Here, y is a simple integer from 0 to $N-1$ but $|y\rangle$ is the qubit state representing that integer. So for the scenario of 4 qubits, $y = 5$ corresponds to $|y\rangle = |0101\rangle$ since $0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 5$. We can write $|y\rangle$ more explicitly as $|y\rangle = |y_1 y_1 \ldots y_n\rangle$. To get some more intuition for how this operation works, let us compute the QFT of the 1 qubit computational basis vectors $|0\rangle$ and $|1\rangle$.

$$U_{QFT} |0\rangle = \frac{1}{\sqrt{2}} \sum_{y=0}^{1} e^{i\frac{2\pi(0)y}{N}} |y\rangle = \frac{1}{\sqrt{2}} \Big(|0\rangle + |1\rangle \Big) = |+\rangle$$
$$U_{QFT} |1\rangle = \frac{1}{\sqrt{2}} \sum_{y=0}^{1} e^{i\frac{2\pi(1)y}{N}} |y\rangle = \frac{1}{\sqrt{2}} \Big(|0\rangle + e^{i2\pi} |1\rangle \Big) = \frac{1}{\sqrt{2}} \Big(|0\rangle - |1\rangle \Big) = |-\rangle$$

Indeed the QFT procedure gives us the states $|+\rangle$ and $|-\rangle$ for the computaional basis vectors for $n = 1$. More generally, QFT will transform a set of computational vectors into another, special, set of basis vectors which is called the Fourier basis. The Fourier basis for the one qubit system is simply the familiar vectors $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$.

Our final goal for this section is to draft an algorithm for implementing the Quantum Fourier transform, so that we may use its inverse algorithm for our Phase Estimation procedure in the next section. In order to come up with this algorithhm, we will first simplify the expression in equation (11) into a more explicit form. More specifically, we want to write it as the following explicit tensor product of different states so that we know exactly how each qubit's state changes:

$$\frac{1}{2^{\frac{n}{2}}} \Big(|0\rangle + e^{i\frac{2\pi x}{2}} |1\rangle \Big) \otimes \Big(|0\rangle + e^{i\frac{2\pi x}{4}} |1\rangle \Big) \otimes \ldots \Big(|0\rangle + e^{i\frac{2\pi x}{2^n}} |1\rangle \Big) \tag{12}$$

Let us start with the fact that y can be written as $y = \sum_{k=1}^{n} 2^{n-k} y_k$. This comes simply from how we find the value y from its binary digits $|y_1 y_2 \ldots y_n\rangle$.

$$U_{QFT} |x\rangle = \frac{1}{2^{\frac{n}{2}}} \sum_{y=0}^{2^n-1} e^{i\frac{2\pi x}{N} \sum_{k=0}^{n} 2^{n-k} y_k} |y\rangle$$

Now, we do 4 things. Firstly, thanks to exponential laws, we can write the exponential in the expression as a product of exponentials.

$$e^{i\frac{2\pi x}{N}\sum_{k=1}^{n}2^{n-k}y_k} = \prod_{k=1}^{n} e^{i\frac{2\pi x}{2^n}2^{n-k}y_k} = \prod_{k=1}^{n} e^{i\frac{2\pi x}{2^k}y_k}.$$

Secondly, We can rewrite $|y\rangle$ as its tensor product

$$|y\rangle = |y_1 y_2 \dots y_n\rangle = |y_1\rangle \otimes |y_2\rangle \dots \otimes |y_n\rangle = \bigotimes_{k=1}^{n} |y_k\rangle .$$

Thirdly, now that everything is in terms of the binary values y_k, instead of summing on the values of y, we can run n sums, one for every coefficient y_k.

$$\frac{1}{2^{\frac{n}{2}}}\sum_{y=0}^{2^n-1}\Big(\prod_{k=1}^{n} e^{i\frac{2\pi x}{2^k}y_k}\bigotimes_{k=1}^{n}|y_k\rangle\Big) = \frac{1}{2^{\frac{n}{2}}}\sum_{y_1=0}^{1}\sum_{y_2=0}^{1}\dots\sum_{y_n=0}^{1}\Big(\prod_{k=1}^{n} e^{i\frac{2\pi x}{2^k}y_k}\bigotimes_{k=1}^{n}|y_k\rangle\Big)$$

Fourthly, and finally, we can combine the product and the tensor product into just a tensor product. To accomplish this we group terms that correspond to the same coefficient of $|y\rangle$, say y_k. Them we find that

$$\prod_{k=1}^{n} e^{i\frac{2\pi x}{N}2^{n-k}y_k}\bigotimes_{k=1}^{n}|y_k\rangle = \bigotimes_{k=1}^{n} e^{i\frac{2\pi x}{N}2^{n-k}y_k}|y_k\rangle$$

Combining all this, we find that

$$\begin{aligned}
U_{QFT}|x\rangle =& \frac{1}{2^{\frac{n}{2}}}\sum_{y=0}^{2^n-1} e^{i\frac{2\pi x}{N}\sum_{k=1}^{n}2^{n-k}y_k}|y\rangle & (13)\\
=& \frac{1}{2^{\frac{n}{2}}}\sum_{y=0}^{2^n-1}\Big(\prod_{k=1}^{n} e^{i\frac{2\pi x}{2^k}y_k}|y\rangle\Big) & (14)\\
=& \frac{1}{2^{\frac{n}{2}}}\sum_{y_1=0}^{1}\sum_{y_2=0}^{1}\dots\sum_{y_n=0}^{1}\Big(\prod_{k=1}^{n} e^{i\frac{2\pi x}{2^k}y_k}\bigotimes_{k=1}^{n}|y_k\rangle\Big) & (15)\\
=& \frac{1}{2^{\frac{n}{2}}}\sum_{y_1=0}^{1}\sum_{y_2=0}^{1}\dots\sum_{y_n=0}^{1}\Big(\bigotimes_{k=1}^{n} e^{i\frac{2\pi x}{2^k}y_k}|y_k\rangle\Big) & (16)\\
=& \frac{1}{2^{\frac{n}{2}}}\bigotimes_{k=1}^{n}\Big(\sum_{y_1=0}^{1}\sum_{y_2=0}^{1}\dots\sum_{y_k=0}^{1}\dots\sum_{y_n=0}^{1} e^{i\frac{2\pi x}{2^k}y_k}|y_k\rangle\Big) & (17)\\
=& \frac{1}{2^{\frac{n}{2}}}\bigotimes_{k=1}^{n}\sum_{y_k=0}^{1} e^{i\frac{2\pi x}{N}2^{n-k}y_k}|y_k\rangle = \boxed{\frac{1}{2^{\frac{n}{2}}}\bigotimes_{k=1}^{n}\Big(|0\rangle + e^{i\frac{2\pi x}{2^k}y_k}|1\rangle\Big)} & (18)
\end{aligned}$$

To explain what happened in equations (16) through to equation (18): We switched the order of summation and the tensor product, and then found that the expression being summed only depended on y_k and not on any of the other coefficients $y_1, y_2 \dots y_{k-1}, y_{k+1} \dots y_n$, and hence we were able to get rid of all but the y_k sum.

The final result in equation (18) is really nice, because each state in the tensor product corresponds to the superposition that the qubits in our system were transformed into. If we expand the notation in (18) out to show each state explicitly then we get

$$\frac{1}{2^{\frac{n}{2}}}\Big(|0\rangle + e^{i\frac{2\pi x}{2}} |1\rangle \Big) \otimes \Big(|0\rangle + e^{i\frac{2\pi x}{4}} |1\rangle \Big) \otimes \dots \Big(|0\rangle + e^{i\frac{2\pi x}{2^n}} |1\rangle \Big) \tag{19}$$

There is one, final, simplification we can make to equation (19). Specifically, we can write x as a binary string $x_1x_2\dots x_n$. We know the mapping of this to the integer x as discussed before. Binary representation can also be used to write decimal numbers. For example, $.x_1x_2\dots x_n$ means

$$x_1 \cdot 2^{-1} + x_2 \cdot 2^{-2} + \dots x_n \cdot 2^{-n} = x_1 \cdot \frac{1}{2} + x_2 \cdot \frac{1}{4} + \dots x_n \cdot \frac{1}{2^n}$$

The reason this simplification is important is that it allows us to write the exponentials in equation (19) in terms of the specific coefficients x_k as follows:

$$e^{i\frac{2\pi}{2^k}x_1x_2\dots x_n} = e^{i2\pi x_1x_2\dots x_{n-k}.x_{n-k+1}\dots x_n} = e^{i2\pi x_1x_2\dots x_{n-k}} e^{i2\pi(0.x_{n-k+1}x_{n-k+2}\dots x_n)} \tag{20}$$

$$= e^{i2\pi(0.x_{n-k+1}x_{n-k+2}\dots x_n)} \tag{21}$$

I was able to ignore the coefficients $x_1x_2\dots x_{n-k}$ when going from (20) to (21) because that was the integer part of the string and we know that $e^{i2\pi n} = 1$ for any integer n. Now our tensor product is even more well specified as

$$\frac{1}{2^{\frac{n}{2}}}\Big(|0\rangle + e^{i2\pi(0.x_n)} |1\rangle \Big) \otimes \Big(|0\rangle + e^{i2\pi(0.x_{n-1}x_n)} |1\rangle \Big) \otimes \dots \Big(|0\rangle + e^{i2\pi(0.x_1x_2\dots x_n)} |1\rangle \Big) \tag{22}$$

Recall that our original state before the transform was

$$|x\rangle = |x_1x_2\dots x_n\rangle = |x_1\rangle \otimes |x_2\rangle \otimes \dots |x_n\rangle .$$

This means that the k^{th} qubit in our system is transformed in the following way:

$$|x_k\rangle \xrightarrow{\text{QFT}} \frac{1}{\sqrt{2}}\Big(|0\rangle + e^{i2\pi(0.x_{n-k+1}x_{n-k+2}\dots x_n)} |1\rangle \Big) \tag{23}$$

$$= \frac{1}{\sqrt{2}}\Big(|0\rangle + e^{i2\pi\frac{x_{n-k+1}}{2}} e^{i2\pi\frac{x_{n-k+2}}{4}} \dots e^{i2\pi\frac{x_n}{2^k}} |1\rangle \Big) \tag{24}$$

You should notice that $|x'_k\rangle$ is very similar to the result of a Hadamard gate on a state $|u\rangle$, $u = \{0, 1\}$:

$$|x'_k\rangle = \frac{|0\rangle + e^{i2\pi 0.x_k} |1\rangle}{\sqrt{2}}. \tag{25}$$

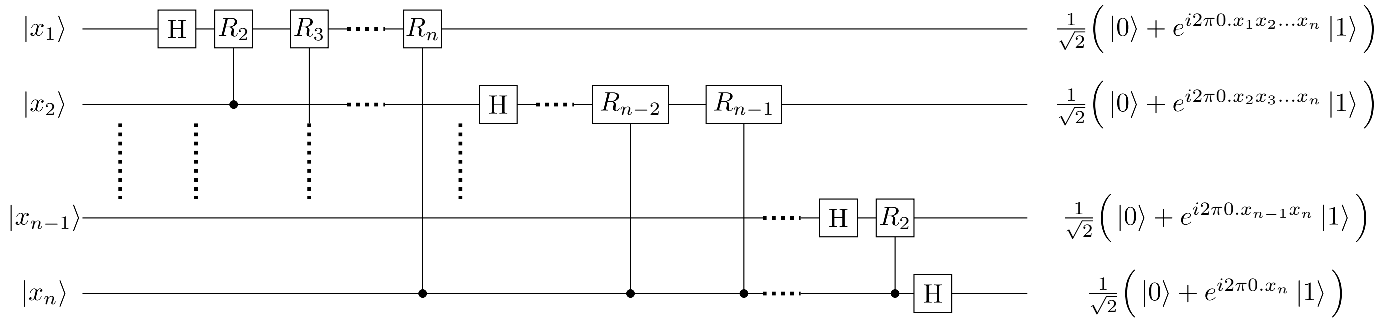
$|x_1\rangle$
$|x_2\rangle$
$|x_{n-1}\rangle$
$|x_n\rangle$
H
R_2
R_3
R_n
R_{n-2}
R_{n-1}
$\frac{1}{\sqrt{2}}\left(|0\rangle + e^{i2\pi 0.x_1x_2\dots x_n}|1\rangle\right)$
$\frac{1}{\sqrt{2}}\left(|0\rangle + e^{i2\pi 0.x_2x_3\dots x_n}|1\rangle\right)$
$\frac{1}{\sqrt{2}}\left(|0\rangle + e^{i2\pi 0.x_{n-1}x_n}|1\rangle\right)$
$\frac{1}{\sqrt{2}}\left(|0\rangle + e^{i2\pi 0.x_n}|1\rangle\right)$

Figure 4: Circuit to implement the Quantum Fourier Transform

The difference is there are also phases $e^{i2\pi\phi}$ that are being thrust onto the $|1\rangle$ term in equation (24). Thus, our strategy to finally draft an algorithm to implement the Quantum Fourier Transform will be to first pass every qubit through a Hadamard gate to get the result of equation (25), and then add additional phases to only the $|1\rangle$ term to bring each qubit into the necessary state. This second step may be accomplished by introducing a new unitary gate:

$$R_k = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{2\pi}{2^k}} \end{bmatrix} \tag{26}$$

$$R_k|0\rangle = |0\rangle \quad R_k|1\rangle = e^{i\frac{2\pi}{2^k}}|1\rangle$$

Specifically, we will be using the controlled R_k gate. If we pass the original value of the x_k bit into R_k as the control value then it creates just the right phase. When $x_k = 0$ we get

$$R_k[0]|1\rangle = |1\rangle$$

which makes sense since $e^{i\frac{2\pi x_k}{2^k}} = e^{i\frac{2\pi(0)}{2^k}} = 1$. Similarly, when $x_k = 1$, we get the phase we are looking for:

$$R_k[1]|1\rangle = e^{i\frac{2\pi x_k}{2^k}}|1\rangle$$

Now all that is left for us is to use these ideas in the QFT circuit in figure **??** to produce the states in equation (24) . Let us analyse just the first line of the circuit, where the Hadamard gate followed by the first controlled gate produces

$$R_2[x_2]\frac{|0\rangle + e^{i2\pi 0.x_1}|1\rangle}{\sqrt{2}} \otimes |x_2x_3\dots x_n\rangle = \frac{|0\rangle + e^{i2\pi 0.x_1x_2}|1\rangle}{\sqrt{2}} \otimes |x_2x_3\dots x_n\rangle \tag{27}$$

The following controlled R_k gates have a similar effect, adding more bits to the phase, finally giving us the state

$$\frac{|0\rangle + e^{i2\pi 0.x_1x_2\dots x_n}|1\rangle}{\sqrt{2}} \otimes |x_2x_3\dots x_n\rangle \tag{28}$$

Then, you can see that the second line of the circuit produces the appropiate state for the second qubit:

$$\frac{1}{2}\Big(|0\rangle + e^{i2\pi 0.x_1x_2\ldots x_n} |1\rangle \Big) \otimes \Big(|0\rangle + e^{i2\pi 0.x_2x_3\ldots x_n} |1\rangle \Big) \otimes |x_3 \ldots x_n\rangle \tag{29}$$

Continuing on for all n lines of the circuit, we finally arrive at the final output state of the qubits:

$$\frac{1}{2^{\frac{n}{2}}}\Big(|0\rangle + e^{i2\pi 0.x_1x_2\ldots x_n} |1\rangle \Big) \otimes \Big(|0\rangle + e^{i2\pi 0.x_2x_3\ldots x_n} |1\rangle \Big) \otimes \ldots \Big(|0\rangle + e^{i2\pi 0.x_n} \Big) \tag{30}$$

Comparing this to the states in equation (22), we see that quation (30) has the same qubit states as it, but in reverse order. This is not a big deal as there exist swap operations that can correct the ordering and give us our desired result without adding any significant computational time. This swap operation is the final step in our QFT procedure

9 Quantum Phase Estimation

Now we are very well acquainted with the Quantum Fourier transform and its implementation on a Quantum computer. I made the claim that there exists a Quantum algorithm that can transform n bits into the set of states in equation (9). This set of states is completely analogous to the Fourier basis for n qubits.

Let us start this section with constructing the algorithm to produce the states in equation (9) as I promised to do in the previous section.

The main gates we will use for the circuit will be controlled U^{2^k} gates where U^{2^k} is simply the U gate applied 2^k times. Given that the control bit is 1, this gate will apply the unitary transformation U. Recall that our whole goal is to find eigenvalues for this general unitary matrix U.

We use two different groups of qubits for this circuit, commmonly referred to as 'registers' of qubits. The first register has n qubits which serve as the control bits for the controlled U^{2^k} gates and are transformed into the form in equation (9) through phase kick back as will be shown soon. These n bits will eventually represent our approximation for ϕ at the end after we apply an inverse QFT. The second register is the set of qubits that represent the eigenstate $|u\rangle$. They are our target bits. When the control bit equals 1, the unitary gates produce a the phase we are looking to produce.

Let us analyse a general line of the circuit in Figure 5. to see how phase kickback occurs. Specifically,

$$\begin{aligned} U^{2^{k-1}}\Big[\frac{|0\rangle + |1\rangle}{\sqrt{2}}\Big] |u\rangle &= \frac{1}{\sqrt{2}}\Big(U^{2^{k-1}}[|0\rangle] |u\rangle + U^{2^{k-1}}[|1\rangle] |u\rangle \Big) \\ &= \frac{1}{\sqrt{2}}\Big(|0\rangle \otimes |u\rangle + |1\rangle \otimes e^{i2\pi 2^{k-1}\phi} |u\rangle \Big) \\ &= \frac{1}{\sqrt{2}}\Big(|0\rangle + e^{i2\pi 2^{k-1}\phi} |1\rangle \Big) \otimes |u\rangle . \end{aligned} \tag{31}$$

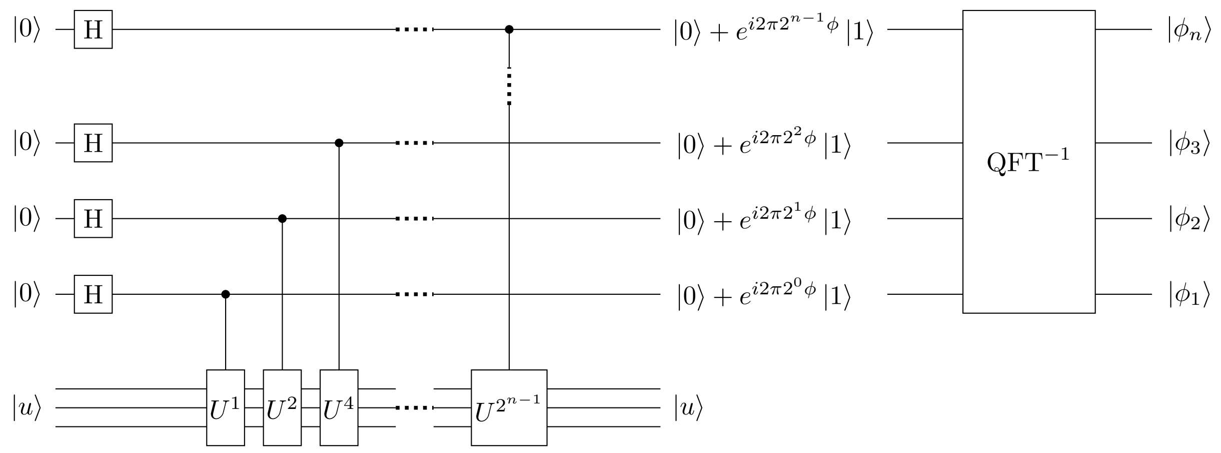
$|0\rangle$
H
$|0\rangle + e^{i2\pi 2^{n-1}\phi}|1\rangle$
$|\phi_n\rangle$
$|0\rangle + e^{i2\pi 2^{2}\phi}|1\rangle$
QFT^{-1}
$|\phi_3\rangle$
$|0\rangle + e^{i2\pi 2^{1}\phi}|1\rangle$
$|\phi_2\rangle$
$|0\rangle + e^{i2\pi 2^{0}\phi}|1\rangle$
$|\phi_1\rangle$
$|u\rangle$
U^1
U^2
U^4
$U^{2^{n-1}}$

Figure 5: Circuit to implement Quantum Phase Estimation

Hence the first register is transformed into

$$\frac{1}{2^{\frac{n}{2}}}\Big(|0\rangle + e^{i2\pi 2^{n-1}\phi}|1\rangle\Big)\otimes\Big(|0\rangle + e^{i2\pi 2^{n-2}\phi}|1\rangle\Big)\otimes\dots\Big(|0\rangle + e^{i2\pi 2^{1}\phi}|1\rangle\Big)\otimes\Big(|0\rangle + e^{i2\pi 2^{0}\phi}|1\rangle\Big) \tag{32}$$

Similarly to x in the previous section, it is useful to write ϕ as a decimal binary expansion. We can say that

$$\phi = 0.\phi_1\phi_2\dots\phi_n$$

You may wonder why we are only including decimal bits. Any bits on the left hand side of the decimal point has no real effect on the phase. Recall that our eigenvalue is $e^{i2\pi\phi}$. If we consider bits to the left of the decimal point, then we quite simply find that

$$\begin{aligned} e^{i2\pi\phi_{-m}\dots\phi_{-1}\phi_0.\phi_1\phi_2\dots\phi_n} =& e^{i2\pi\phi_{-m}\dots\phi_{-1}\phi_0}e^{i2\pi 0.\phi_1\phi_2\dots\phi_n} \\ =& e^{i2\pi 0.\phi_1\phi_2\dots\phi_n} \end{aligned}$$

With this in mind, our tensor product in equation (32) becomes

$$\begin{aligned} &\frac{1}{2^{\frac{n}{2}}}\Big(|0\rangle + e^{i2\pi\phi_1\phi_2\dots\phi_{n-1}.\phi_n}|1\rangle\Big)\otimes\Big(|0\rangle + e^{i2\pi\phi_1\phi_2\dots\phi_{n-2}.\phi_{n-1}\phi_n}|1\rangle\Big)\otimes\dots\Big(|0\rangle + e^{i2\pi 0.\phi_1\phi_2\dots\phi_n}|1\rangle\Big) \\ =&\frac{1}{2^{\frac{n}{2}}}\Big(|0\rangle + e^{i2\pi 0.\phi_n}|1\rangle\Big)\otimes\Big(|0\rangle + e^{i2\pi 0.\phi_{n-1}\phi_n}|1\rangle\Big)\otimes\dots\Big(|0\rangle + e^{i2\pi 0.\phi_1\phi_2\dots\phi_n}|1\rangle\Big) \end{aligned} \tag{33}$$

As promised, this is exactly analogous to the result of the Quantum Fourier Transform as seen in equation (22). Thus, in order to recover an approximation for the value ϕ, the final step is to take the inverse of the QFT procedure. Since QFT is a unitary operation, it definitely has an inverse, and the circuit for this inverse can be contructed without much trouble by reversing the original QFT circuit. The final result

$$|\phi_1\rangle\otimes|\phi_2\rangle\dots\otimes\phi_n$$

is a definite state of n qubits and does not contain any superpositions meaning measuring the state simply gives us the string $\phi_1\phi_2\dots\phi_n$ which represents the phase value

$$\phi \approx 0.\phi_1\phi_2\phi_3\dots\phi_n \tag{34}$$

I have repeatedly referred to this result as an approximation to ϕ rather than its exact value, so let us end this section off with why this is an approximation. The issue lies in the fact that we are trying to approximate ϕ with the finite number of qubits n. Not only is it possible that the representation of the true value of ϕ may require more than n bits, it is possible that ϕ is irrational with infinite bits in its representation. The best we can do for a longer string ϕ while using only n qubits is get within an error of $|\phi_{actual} - \phi_{approximated}| < \frac{1}{2^n}$. Luckily for us, though, if you take a reasonably large number of qubits, there is a large probability that we get a good approximation for ϕ with error of the order of $\frac{1}{2^n}$. This is a fact that I will not attempt to prove here, but a deep discussion of it can be found in Nielson's Quantum Computation Unit 5.2.1.

10 Order Finding allows Prime Factorization

Now we are finally ready for the heart of this paper's discussion; factoring numbers on a Quantum computer. In his 1994 paper, Peter Shor outlined a Quantum algorithm for his task. He utilised the strong link between prime factorization and the problem of finding the order of a number.

DEFINITION 1. Suppose you have two positive integers, x and N such that x and N are co-prime, or in other words $\gcd(x, N) = 1$. The order r for these two integers is defined as the smallest positive integer such that $x^r \equiv 1 \pmod{N}$.

This definition is so important to us because the problem of factorization can be reduced to order finding. Suppose you want to prime factorize a number N. Before discussing how order finding helps in factorization, however, we should cover some simple cases. For example, if N is even, it is an easy task to reduce it to find its even factors since we know they are of the form 2^k and division by 2 is not a difficult task for classical computers. For the following discussion, let us just assume that we have found the even factors, and our N is odd. Next, it is possible that $N = p^n$, where p is prime. This is the case of N having just one distinct prime factor. There is a simple classical algorithm to find p if this is true. We simply cycle through integers $1 \leq n \leq \log_2 N$ and compute the quantity $\sqrt[n]{N}$. If we find that $p = \sqrt[n]{N}$ is an integer for any choice of n, we know that $N = p^m$. Once again, taking the n^{th} root of a number is not particularly difficult computationally, and checking for such a factor before starting the order finding procedure will not cost us any significant time.

Now to see how order finding relates to factoring N, we can start by finding a number $0 < x < N$ which is co-prime to N. Then we may find the order such that $x^r = 1 \pmod{N}$. We can express this modular relation instead using explicit constants for the quotient and remainder.

$$\begin{aligned} x^r &= kN + 1 \\ x^r - 1 &= kN \\ (x^{\frac{r}{2}} - 1)(x^{\frac{r}{2}} + 1) &= kN \end{aligned} \tag{35}$$

This means that $(x^{\frac{r}{2}}-1)(x^{\frac{r}{2}}+1)=0 \pmod N$, which implies that there exists a $\gcd(x^{\frac{r}{2}}-1,N)$ which is a factor of N! Thus, if we can find a co-prime to N, and compute its order, we can find ourselves a nontrivial divisor of N. However, thereer, there are some possibilfailure thatf failure that wewith.with. Firstly $x^{\frac{r}{2}}-1$ is an integer only if $\frac{r}{2}$ is also an integer. Hence, the order r we find must be even. Secondly, there is the case where $x^{\frac{r}{2}}=-1 \pmod N$. This is a problem because it implies that $x^{\frac{r}{2}}+1=0 \pmod N$ and $x^{\frac{r}{2}}-1=-2 \pmod N$. You can see the original discussion of order finding as a path to factoring in Gary Miller's 1976 paper in [Mil76].

Luckily, these two possibilities do not pose a big issue for us because for a chosen coprime x, we can prove that there is a high probability that its order r with N is even and that $x^{\frac{r}{2}} \neq -1 \mod N$.

THEOREM 2. *Suppose a number $N=p_1^{s_1}p_2^{s^2}\dots p_n^{s_n}$, and a randomly chosen number x which is coprime to it. Then their order r satisifes the conditions that r is even and $x^{\frac{r}{2}} \neq 0 \mod N$ with the probability*

$$P\big(r\ is\ even, x^{\frac{r}{2}} \neq 0 \pmod N\big) \geq 1-\frac{1}{2^n} \tag{36}$$

There exists an efficient algorithm for computing the greatest common denominators of two numbers (see [Knu98]), and hence finding $\gcd(x^{\frac{r}{2}}-1,N)$ is not a bottleneck for our procedure either.

11 The Order Finding Algorithm

Now we know that if we can find the order of N and one of its coprimes x, then there is a high probability this will allow us to find a factor of N. This all is only of any practical value to us, however, if we can actually efficiently find the required order r. The final piece in the puzzle is the fact that we can construct an efficient Quantum algorithm for this task.

As eluded to earlier, the order finding algorithm turns out to simply be the phase estimation algorithm, applied to a special unitary operator.

$$U|y\rangle = |xy \pmod N\rangle \tag{37}$$

Note that

$$U|x^k \pmod N\rangle = |x^{k+1} \pmod N \pmod N\rangle = |x^{k+1} \pmod N\rangle$$

which allows us to find an eigenstate for U. Specifically any state of the form

$$|u_s\rangle = \frac{1}{\sqrt{r}}\sum_{k=0}^{r-1} e^{i\frac{2\pi sk}{r}}|x^k \bmod N\rangle \tag{38}$$

where $0 < s < r-1$ is an integer is an eigenstate of U as

$$\begin{aligned}
U\left|u_s\right\rangle =& \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-i\frac{2\pi sk}{r}} \left|x^{k+1} \bmod N\right\rangle \\
=& \frac{1}{\sqrt{r}} \sum_{k'=1}^{r} e^{-i\frac{2\pi s(k'-1)}{r}} \left|x^{k'} \bmod N\right\rangle & (39)\\
=& \frac{1}{\sqrt{r}} \sum_{k'=0}^{r-1} e^{-i\frac{2\pi s(k'-1)}{r}} \left|x^{k'} \bmod N\right\rangle & (40)\\
=& e^{i\frac{2\pi s}{r}} \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-i\frac{2\pi sk}{r}} \left|x^{k} \bmod N\right\rangle = e^{i\frac{2\pi s}{r}} \left|u_s\right\rangle & (41)
\end{aligned}$$

Between equations (39) and (40), I used the fact that

$$e^{-i\frac{2\pi s}{r}(r-1)} \left|x^r \bmod N\right\rangle = e^{-i\frac{2\pi sr}{r}} e^{i\frac{2\pi s}{r}} \left|1 \bmod N\right\rangle = e^{i\frac{2\pi s}{r}} \left|1 \bmod N\right\rangle$$

Now there is only one thing left to do. We apply the phase estimation procedure to the unitary gate represented in equation (37) and the we hence compute, to a high degree of accuracy, an estimation for the value $\phi \approx \frac{s}{r}$.

12 Shor's Factoring Algorithm

With all this we are finally ready to state the procedure for factoring a general composite integer N on a Quantum algorithm. Let's state the procedure as a series of steps.

1. If N is even, return 2 as a factor.
2. We check if N is of the form $N = p^m$ using the classical algorithm mentioned prior. If this is found to be true, we return p and end our algorithm.
3. Choose a random integer in the range $1 < x < N$. If $\gcd(x, N) > 1$, then we have hit the jackpot as the result of this GCD, in fact, gives us a non-trivial factor of N, and we can return it. If, instead, $\gcd(x, N) = 1$, then x is co-prime to N and we proceed to Step 4.
4. We now find the order r of $x \mod N$ using the order finding algorithm discussed prior.
5. We check if r is even and if $x\frac{r}{2} \neq -1 \mod N$. If these conditions hold, we can compute $\gcd(x^{\frac{r}{2}} - 1, N)$ and divide N by it to test if it is a non-trivial factor of N. If it is, then we return this as a factor. However, if either condition does not hold, our algorithm has failed. We must try again with a new choice of x.

References

[DJ92] David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proc. R. Soc. Lond., Ser. A*, 439(1907):553–558, 1992.

[Knu98] Donald E. Knuth. *The art of computer programming. Vol. 2: Seminumerical algorithms.* Bonn: Addison-Wesley, 3rd ed. edition, 1998.

[Mil76] Gary L. Miller. Riemann's hypothesis and tests for primality. *J. Comput. Syst. Sci.*, 13:300–317, 1976.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997.