Global Journal of Computer Science and Technology: G Interdisciplinary
Volume 17 Issue 2 Version 1.0 Year 2017
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

Quantum Computing Tutorial Bits vs Qubits and Shor's Algorithm

By Koffka Khan
The University of the West Indies

Abstract- The speculative inquiry that computation could be done in general more efficiently by utilizing quantum effects was introduced by Richard Feynman. Peter Shor described a polynomial time quantum algorithm for factoring integers by a quantum machine, which proved the speculation true. Quantum systems utilize exponential parallelism, which cannot be done by classical computers. However, quantum decoherence poses a difficulty for measuring quantum states in modern quantum computers. This paper elaborates on some basic concepts applied to quantum computing. It first outlines these key concepts, introduces the mathematics needed for understanding quantum computing and finally explores the Shor's Algorithm as it applies to both classical and quantum computer security.

Keywords: *quantum; computing; shor's; algorithm; security.*

GJCST-G Classification: *I.1.2, F.2.2*

Strictly as per the compliance and regulations of:

© 2017. Koffka Khan. This is a research/review paper, distributed under the terms of the Creative Commons Attribution-Noncommercial 3.0 Unported License http://creativecommons.org/ licenses/by-nc/3.0/), permitting all non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

Quantum Computing Tutorial Bits vs Qubits and Shor's Algorithm

Koffka Khan

Abstract- The speculative inquiry that computation could be done in general more efficiently by utilizing quantum effects was introduced by Richard Feynman. Peter Shor described a polynomial time quantum algorithm for factoring integers by a quantum machine, which proved the speculation true. Quantum systems utilize exponential parallelism, which cannot be done by classical computers. However, quantum decoherence poses a difficulty for measuring quantum states in modern quantum computers. This paper elaborates on some basic concepts applied to quantum computing. It first outlines these key concepts, introduces the mathematics needed for understanding quantum computing and finally explores the Shor's Algorithm as it applies to both classical and quantum computer security.

Keywords: quantum; computing; shor's; algorithm; security.

I. Introduction

In 2017, IBM has a 16-qubit Quantum computer on the cloud available for users worldwide. These and other revolutionary breakthroughs over the past years have propelled the world of quantum computing into the spotlight.

First let us see, how classical computers work. A classical computer works with the binary numbering system, and the computer is not able to compute with the decimal numbering system. Binary system has only two digits. All arithmetic operations are done by the binary system based logic. Let us use an example of adding two single digit binary numbers using yes or no logic. Turn the first bit on, if any one of the bit is on, that is exclusive OR. Turn the second bit on if both the bits are on, that is AND. We can use electrical-switches as an input device and lights as output device. Transistors can be used for binary-logic based operations and turning on or off the lights based on the switch settings. Transistors can be inter-connected in particular way to pass the electric-current by with switches. A mobile phone has millions of transistors inside. A computer has billions of transistors inside. Computer likes binary states.

How about the quantum state, which has the states of both 0 and 1 at the same time? Binary bit state may be 0 or 1. Quantum qubit state will be both 0 and 1 at the same time. As individual digits of input numbers have all possible ways, the result will also have all possible values. Two single digit qubit numbers addition will make 4 possible combinations. Two double-digit Qubit numbers addition will make 16 possible combinations and so on. All types of arithmetic operations do this kind of computation. Therefore, a quantum computer computes all possible ways in parallel. But classical computer computes only one at a time. Take the maze as an example. The maze has an entrance and the maze is inside. The entrance is split into multiple paths and has only one exit. The task of computer is to find the correct path which leads to the exit. Classical computer has to travel each path to find the exit. However, a quantum computer can travel all paths simultaneously and find the exit immediately. It computes all possible combinations simultaneously and choosing the best one.

Classical computer uses transistors to create binary-based Logic-Gates. Subatomic particles such as electrons and photons behave in a very strange way. Electron has a property of spin. The spin state may be Up, Down, Right or left. The spin state will be both Up and Down or Right and Left simultaneously in particular scenario. Such a state is called superposition [3] state. Photon has a property of polarization. The polarization state may be horizontal, or vertical. The polarization state will be both horizontal and vertical simultaneously in particular scenario. Light has strange behavior in the double-slit experiment. Light without any slit shows normal pattern. Light passing through single slit is spread out, because of quantum uncertainty behavior. Light passing through double slit shows interference pattern because of the wave behavior of light. Passing single photon at a time in single slit hits random place, accumulates and shows the same spread pattern over the period. Passing single photon at a time in double slit hits random place, accumulates and shows the same interference pattern over the period. How can a single photon which is not a wave, show interference pattern? Actually, it splits in to two photons, passing through slits simultaneously, interferes with itself and shows interference pattern. The photon is in superposition state of passing both slits. The spread-pattern of single slit is also the superposition state of a photon is in all position simultaneously. The photon resides in this area with the possibility of all combinations. This superposition state can be used to create qubits, which is used in quantum computers. Superposition of particle spin can be used to create quantum logic gates. The superposition is collapsed and turned in to definite state when it gets measured.

Author: Department of Computing and Information Technology, University of the West Indies, Trinidad and Tobago, W.I.
e-mail: koffka_@hotmail.com

© 2017 Global Journals Inc. (US)

This paper consists of three sections. Section II discusses Quantum Concepts. Section III explores the Mathematics that support Quantum Computing, Section IV explains why cryptographic codes are so hard to break and finally Section V discusses Shor's Algorithm and Quantum Security.

II. Quantum Concepts

The fundamental unit of a classical computer is a bit. Bits have two states, 0 and 1. A classical computer takes in a string of bits and use logic gates to switch some of the bits. Quantum computers use quantum bits (qubits, [6]). Like a bit, a qubit can be in state 0 or state 1. Also like a classical computer, the initial program for a quantum computer is just a string of zeros and ones. However, while a quantum computer is running, its qubits can also be in infinitely many super positions [3] between 0 and 1. When a qubit is in a superposition, it has some probability of being in state 0 and some probability of being in state 1. You can think of a superposition as being a mixed state partway between 0 and 1. However, super positions are fragile. If we look at it or try to measure it, the qubit will collapse into a basic state, either 0 or 1. You might know this from the famous Schrodinger's cat thought experiment. Before opening the box, the mythic cat is in a superposition of alive and dead. However, when you observe the cat, it is forced to pick a state, alive or dead, not both. Qubit materials are usually things like electrons, where spin up corresponds to state 0 and spin down corresponds to state 1.

Let us see an example of a quantum computation with two qubits. There is four basic states, 0 0, 1 0, 0 1, and 1 1. The two classical bits can be in these states. However, there are also infinitely many states formed by superpositions or combinations of these basic states. Each operation of a quantum computation is performed by a quantum gate, which, like a classical gate, changes the state the qubits are in. Let us start our quantum computation in 0 0 and then apply a quantum gate. Now the qubits are in a superposition. There is a 1/2 probability or 50% chance of being 0 1 and a 1/2 probability of being 1 0. The particular superposition position it is in is a result of the quantum gate we chose to apply. Here is one more quantum gate, changing the state of our computation. At the end of the quantum computation, we observe or measure the system.

However, we cannot see these delicate superpositions. Remember, a superposition is like a mix between basic states. When you observe the computation and look at it from the perspective of these basic states, it must pick one, collapsing the wave function and revealing a single basic state. In this case, it collapsed to state 0 1. If you run the same computation repeatedly, the result will be 0 1 half the time, it will be 1 0 1/6 of the time, and 1 1 1/3 of the time. That is what the numbers in the superposition tell you. The probability that the superposition will collapse into each basic state. So if you run the computation 100 times, roughly 50 times it'll result in the state 0 1, 17 times it will result in state 1 0, and 33 times it will result in state 1 1. This allows you to recover the probabilities and therefore the final superposition of the computation. This does not seem very efficient with two qubits. Nevertheless, as we will see later, it can save you a lot of time with more qubits.

III. Mathematics that Support Quantum Computing

A vector can be a abstract concept in mathematics. Let us define a vector as a list of numbers and the dimension of that vector is the number of numbers in the list. Actual qubits use negative or even complex numbers, but let us deal with non-negative real numbers for now. One qubit is represented as a two-dimensional vector. The state 0, $|0\rangle$ and the state 1, $|1\rangle$. Moreover, this is a superposition, $a|0\rangle + b|1\rangle$. We can visualize the vector on a circle like this. The horizontal component is the square root of the probability of being in state 0. In addition, the vertical component is the square root of the probability of being in state 1. By the Pythagorean Theorem, the length of the vector is 1. Each point on the unit circle is a quantum state. A classical computer can only point up or right, but a quantum computer uses much more of the circle.

What about two qubits? It takes a four-dimensional vector to represent the four possible states. Here is the earlier computation in vector form.

The formula for the length of a two-dimensional vector easily generalizes to the formula for the length of a four-dimensional vector. Therefore, as we said before, all the quantum state vectors have length 1.

The two-dimensional vectors pointed to a spot on the unit circle in two-dimensional space. In addition, these four-dimensional vectors point to a spot on the unit sphere in four-dimensional space, which makes it very hard to visualize. If you have N qubits, there are two to the *N* basic states.

Therefore, a vector on a sphere represents the quantum state in two to the *N* dimensional space. Quantum gates change the system's state. Therefore, they move the state vector around the sphere. Mathematically, this is represented with a unitary matrix. For our purpose, a matrix, specifically a unitary matrix, is a block of numbers that describes how vectors move around the sphere. When we multiply it by the starting vector, 1 0 0 0, we get back a new vector, which represents our second state. Each quantum gate is a different unitary matrix, changing the vector, which represents the state of the qubits. We just apply this quantum gate to the state 0 0, represented by this vector, and got this state as a result. However, if we apply the same gate to state 1 0, represented by this vector, we get this state as a result. Note that the

© 2017 Global Journa ls Inc. (US)

superposition has negative numbers in it. To get the probability that the qubit collapses into each basic state, we just take the absolute value of the numbers. In fact, not only can these numbers be negative, they can actually be complex numbers. Notice that the state of *N* qubits is actually represented on a sphere in two to the *N* complex dimensions, which has twice the dimensionality of the sphere in two to the N real dimensions.

IV. Cryptography

Cracking open secure messages would be easy if only you knew how to factor huge numbers. One of the main methods of cryptography, the encoding and decoding secure communications, uses big prime numbers. It is easy for a computer to find big prime numbers and multiply them together, but it is hard for a computer to do the opposite-- find the prime factors of a big number. The prime factors of a number are all the prime numbers that evenly divide it. Normally, RSA (Rivest Shamir Adleman) [1] cryptography uses these prime factors like keys to decrypt messages. So if you want to eavesdrop, you'll need to find one of these keys to hack in--that is, you'll need to find the prime factors of a big number, and we're talking really big, as in hundreds of digits long. Let us try a small example. What are the prime factors of 35? Well, they are 5 and 7. How did you figure that out? Probably just by looking at it, but even if you had forgotten that fact, you could have just checked all the prime numbers smaller than 35. Does two divide it? No. Does 3? No. Does 5? Yes. And so on. This is for a computer, very time consuming. We will need to do something strategic to factor big numbers. Along with many, many other things Euler thought a lot about prime numbers, relatively prime numbers, and modular arithmetic, which is basically all the math underlying RSA cryptography. Therefore, it makes sense that we would use similar math to break the algorithm. Modular arithmetic is what happens when you count in a circle. Counting modulo 5, or mod 5 for short, goes 0, 1, 2, 3 4, 0, 1, 2, 3 4, 0, 1, 2, and so on. We just use the numbers less than 5 on repeat. We tell time mod 12 or mod 24 depending on your convention. This cyclical counting extends to the arithmetic operations. So 1 plus 2 mod 5 is still just 3, but 2 plus 3 mod 5 is 0, and 2 times 3 is 1 mod 5. Another way to think about modular arithmetic is in terms of the remainder when dividing numbers. Therefore, a slightly more formal definition follows. *a* is congruent to *x mod n* means that when we divide *a* by *n* the remainder is *x*. So 2 times 3 is 6, but when we divide 6 by 5, the remainder is 1. Therefore, 2 times 3 mod 5 is 1. Euler noticed something about modular arithmetic and exponentiation. Let us look at the powers of 3-- 3, 9, 27, 81, 243, and so on. In addition, let us look at them all mod 10.

It is easy to figure out what things are mod 10 because it is just the remainder when you divide by 10, which is the ones digit. So mod 10 our sequence is 3, 9, 7, 1, 3, 9, 7, 1, and so on. Let us repeat the same experiment, but instead of looking at the powers of 3 mod 10, let's look at the powers of 2 mod 7. The powers of 2 are 2, 4, 8, 16, 32, 64, and so on. In addition, mod 7 we get 2, 4, 1, 2, 4, 1, and so on. What do you observe? The sequence of powers just gets bigger and bigger, but the modular versions of the sequence cycle repeats. They repeat the same pattern over and over again, and the last digit of that pattern is always 1. As long as *x* and *n* are relatively prime, meaning they share no prime factors, the sequence *x mod N*, *x squared mod N*, *x cubed mod N*, *x to the fourth mod N*, and so on will always have this property. We call the length of the repeating pattern the period. Therefore, the period of 3 mod 10 is 4, and the period of 2 mod 7 is 3. Here is why the period is important. If the period of *x mod N* is some number *r*, then *r* is the smallest number such that *x* to the *r* is congruent to *1 mod n*. For example, 3 to the fourth is congruent to 1 mod 10, but 3 for the first, 3 squared, and 3 cubed are not 1 mod 10, but let's get back to our original goal. What does all this stuff about modular arithmetic, exponentiation, and periods have to do with factoring large numbers? Let us say I give you a number *n*. I tell you *n* equals *p times q* for two prime numbers *p* and *q*, but I do not tell you anything about those primes. Your job is to find them. Here is how you will do it.

Step one- pick any number smaller than n. Let us call the number you selected *a*. Check to make sure that *a* and *n* are relatively prime by computing the greatest common divisor of *a* and *n*. The greatest common divisor of two numbers is the biggest integer that divides them both, so it's 1 if the two numbers are relatively prime. The Euclidean algorithm is a quick and standard way to find the GCD [2] of two numbers. If they have a divisor in common, that is a factor of n, which is what you have been looking for, and you have saved yourself the rest of the steps.

Step two- compute the period of *a mod N*. Let us call it *r*. For the sake of example, let us say you are trying to find the factors of 35. Therefore, *n* equals 35, and you pick *a* equals 8 since its relatively prime to 35. Then with a little computation, we can see that *r* equals 4. To make all the arithmetic work out, we are going to need to divide *r* by 2. Therefore, we need to know that *r* is even. Later on, we will also need to know that *a* to the *r* over 2 plus 1 is not congruent to *0 mod N*. If either of these things fail, we need to pick a different *a* in step one. Luckily, there is at least a 50% chance you will pick a good value for a. So on average, you will not have to try too many times.

For step three, we will have to do some algebra. Let us start with the fact we know. *a* to the *r* is congruent to *1 mod N*, which, subtracting 1, gives the *a* to the *r minus 1* is congruent to *0 mod N*. Saying that something is *0 mod N* is the same as saying that it's a multiple of *N*. Therefore, there must exist some integer *k* such that *a* to

© 2017 Global Journals Inc. (US)

the *r minus 1* equals *k times N*. Since we assumed *r* is an even number, we can rewrite it as a to the *r* over *2 minus 1* times *a* to the *r* over 2 plus 1 equals *kN*. In addition, since *N* equals *pq*, we'll replace it with *pq*. Here is what happens with the example where we are trying to find the factors of 35. Since the period of 8 mod 35 is 4, we have 8 to the fourth is congruent to *1 mod 35*. Therefore, 8 to the fourth minus 1 is congruent to *0 mod 35*. Actually, 8 to the fourth minus 1 is 4,095, but we only care about its value mod 35. We could rewrite this as 8 to the fourth minus 1 equals *k* times 35 for some integer *k*. Again, we could solve for *k* in this case, but it is irrelevant, so I will leave it as a variable. Rewrite this as 8 squared minus 1 times 8 squared plus 1 equals *k* times *p* times *q* where *p* and *q* are the prime factors of 35 that we're searching for.

Step four-I claim that the greatest common divisor of *a* to the *r* over 2 minus 1 and *N* is one of the prime factors. Let us call it *p*, and the greatest common divisor of *a* to the *r* over 2 plus 1 and *N* is the other prime factor. Let us call it *q*. Why? The equation a to the *r* over 2 minus 1 times *a* to the *r* over 2 plus 1 equals *kpq* means that *p* must divide one of the factors on the left and *q* must divide one of the factors on the left, but they cannot divide the same factor since that factor would be divisible by *N*. Why is neither factor divisible by *N*? For one, we assumed a to the *r* over 2 plus 1 is not congruent to 0 mod *N*. For the other, we know *r* is the minimum value of *x* such that *a* to the *x* is congruent to *1 mod N*. So *a* to the *r* over 2 minus 1 is not congruent to *0 mod N*. Since *p* and *q* divide separate factors on the left side of the equation, we can assume *p* divides *a* to the *r* of 2 minus 1 and *q* divides *a* to the *r* over 2 plus 1. Therefore, our formulas work.

Therefore, in our example, *p* is the greatest common divisor of 63 and 35, which is 7. Moreover, *q* is the greatest common divisor of 65 and 35, which is 5, and is correct. In summary, here is the steps.

- Step one- pick a less than *N*.
- Step two- find the period of *a mod N*.
- Step three- check that *r* is even and *a* to the *r* over 2 plus 1 is not congruent to *0 mod N*. If either of these things fail, we need to go back to step one and pick a new value of a.
- Finally, step four-- let *p* equal the GCD of *a* to the *r* over 2 minus 1 and *N*. In addition, let *q* equal the GCD of *a* to the *r* over 2 plus 1 and *N*.

Step two, finding the period, takes a long time-- in fact, an exponentially long time. All the steps besides two are fast. Instead of looking for a needle in a haystack, we reduced the hard part to one step-- finding the period. In addition-- here is the big twist-- period finding is precisely the kind of thing a quantum computer is good at, and on the next section. The four steps we just reviewed are the outline of Shor's algorithm, and next section shows how to use a quantum computer to dramatically speed up step two.

V. Shor's Algorithm and Quantum Security

Remember, popular forms of cryptography work by multiplying together two large prime numbers and using those primes as keys to recover the message. Therefore, to crack the code, we will need to find the prime factors of a big number. However, that would take a classical computer a long time. Way longer than the encrypted information is probably useful for. However, Shor's algorithm [4] allows us to quickly factor large numbers using a quantum computer. Let us see how a classical computer would factor a prime number. What is the most straightforward way it could find the factors of a number *N*? Well, it could check. Is 2 a factor, is 3 factor, is 4 a factor, and so on. However, if *N* is big, this might take many steps. Now, if a quantum computer is just a bunch of classical computers working in parallel, then we could have one computer check if 2 is a factor, another check if 3 is a factor, and so on. Then it would only require two steps. We have split the many steps of a classical computer among the many parallel computations of a quantum computer. Here is the problem. When we say that a quantum computer is a bunch of classical computers working in parallel, what we really mean is that a quantum computer is in a superposition of basic states, which are the kind of states a classical computer could be in.

Remember, a superposition is a combination of basic states and there is some probability associated with observing each of them. To find that probability, you square the amplitude of the number in front of the basic state. Here, we have N basic states and a 1 over *N* probability of being in each state. Therefore, the quantum computer is not actually in all of these states. It is more like the quantum computer has split itself into N different pieces. However, when you measure a quantum computer, that is, ask for the result of a computation, it does not tell you about all *N* pieces it is in. Instead, it will pick a state, each with probability 1 over *N*, and tell you what that state says. You cannot look at the whole thing. Just one random state. That is a problem for us. Only two the *N* states give useful information. That the number of checked was a divisor of *N*. So the vast majority of the time we run the computation, *N* minus 2 over *N* of the time, the result will just tell you that something is not a factor of *N*. That means our algorithm is no more efficient than checking random numbers to see if they are divisors using a classical computer.

To harness the power of quantum computation, we need each of these basic states, the components of the superposition, to be working together. Right now, they are functioning as separate computers individually searching, which is a problem because the quantum computer cannot tell us about all these independent states. However, if there is some kind of underlying structure to the states, we can use that to amplify the states with the correct answers. In this case, the ones

© 2017 Global Journals Inc. (US)

that give the factors of a number. Then when we measure the quantum state, we will have a high probability of ending up with the correct answer. So instead of checking each number smaller than *N* to see if it is a factor, how does Shor's algorithm find the factors? It needs to utilize the properties of its entire superposition, and not just a few of its basic states. To do that, Shor's algorithm actually uses some number theory, to transform the problem of finding the factors of a given number into a problem of finding a different number, the period of a periodic function. Here is the four basic steps that outlines the number theory in Shor's algorithm for finding the two secret prime factors, *p* and *q*, of a given number *N*. That is, *N* is equal to *p* times *q*.

- Step 1, pick a number, *a* less than *n*, at random.
- Step 2, check to make sure it is not a factor of *N*. Step of *a mod N*.
- Step 3, check that *r* is even, and *a* to the *r* over 2 plus 1 is not congruent to *0 mod N*.
- Step 4, let *p* be the GCD of *a* to the *r* over 2 minus 1 and *N*, and *q* be the GCD of *a* to the *r* over 2 plus 1 and *N*. Then you found p and q, the two prime factors of *N*.

However, step 2 is the extremely long step. Remember, *N* is the number we are trying to find the factors of, and *a* is a selected number smaller than *N*. We are trying to find the smallest number *r*, which we call the period, such that a to the *r* is congruent to *1 mod N*. It is easy to find the period of a small example just by checking the powers of *a mod N* until we get 1. So if *N* is equal to 7 and a is equal to 2, we compute 2 to the 1 mod and 2 to the 3 mod 7 is 1. Therefore, the period is 3. However, if *N* is big, then *r*, the period, can be as big as *N*. There is no known efficient classical way to find the period. Remember how we tried to find the factors of *N* by letting the quantum computer act as *N* parallel classical computers, and using each to check a different factor? We could try the same thing to find the period. We begin with *N* different states representing the numbers for each state, we compute *a* to the *x mod N*, where *x* is the number of the state. So now the states are *a* to the 1 mod *N*, *a* to the 2 mod *N*, *a* to the 3 mod *N*, and so on. Then we just look for the smallest one that says 1, and we are done. That is when we run into the same problem as before. We cannot just scan all the states at once. When we look at the result of a quantum computation, it just shows one random state, which is not very helpful.

However, there is something different about this current problem. Something that will possibly help us. The period is a global property of this quantum superposition. It is not just a special fact about one or two of the basic states. It is a fact about this entire wave of numbers created by superposition, how often it repeats. That is the period. We can use this to our advantage. We apply something known as the quantum Fourier transform [5] to the superposition *a* to the 1 mod *N*, *a* to the 2 mod *N*, *a* to the 3 mod *N*, and so on. The quantum Fourier transform utilizes the ideas of quantum physics to do exactly what we want. It uses resonances to amplify the basic state associated with the correct period, and the incorrect answers destructively interfere, which suppress their amplitudes. After applying the quantum Fourier transform, there is a very high probability that we will pick the correct period. So how does it work?

To understand the quantum Fourier transform, we will need to start with a quick version of a branch of math known as complex analysis. What we will really be doing is adding complex roots of unity. However, if you are not familiar with that concept, do not worry. Start with a bunch of circles. On the first, we will put two equally spaced dots. On the next, we will put three equally spaced lines. On the next, four equally spaced dots. And so on. Notice, though, we always put one of the dots on the middle right side, the 0-degree angle. Start a dial on that special point. By the way, these dots are called complex roots of unity. Now, let us focus on the circle with three dots. We will move the dial counter-clockwise through the points. In addition, underneath the dial, we will form a path consisting of arrows where the direction of the arrow is given by the direction in which the dial points. For example, with three dots, the first arrow points east. Then move the dial one dot counterclockwise and connect to the first arrow another that points northwest, like the dial. Move the dial again and connect another arrow pointing southwest, the same direction as the dial. Notice that after three arrows, we are back where we started. This is what it looks like on a circle with six dots. Again, after six arrows, we are back to the starting place. Remember that we have a superposition whose basic states look like *a* to the 1 mod N, *a* to the 2 mod N, *a* to the 3 mod N, and so on. Let us pick a tiny example, like *a* equals 2 and N equals 7. Then the components of the superposition are 2 to the 1 mod 7, 2 to the 2 mod 7, 2 to the 3 mod 7, and so on, which is the repeating pattern 2 4 1, 2 4. Because this example is so small, we can just see that the period is three by looking at it. However, how can we use our dials to figure out period? We will move along the sequence *a* to the 1 mod N, *a* to the 2 mod N, *a* to the 3 mod N. For each term in the sequence, move every dial once counter-clockwise. Any time we encounter a 1, stop and record where the dial is pointing with an arrow. Let us focus on the sequence. The dial with three points is always pointing directly east when we record its values. Therefore, our path of arrows just runs off to the right. However, what happens to the dial with four points? The first time we encounter a 1, its facing south. The next time, it's facing west. The next time, it is facing north. In addition, the fourth time we encounter. Therefore, our path of arrows has looped back to where it started. In fact, this will happen with all of the numbers besides 3. They will all just make loops near the starting

© 2017 Global Journals Inc. (US)

point. The distance of the arrow from the starting point is like the amplitude, or probability of a state. Since we are most likely to observe these states at the end of the computation, we are set. We have magnified the correct answer. In addition, that is roughly how the quantum Fourier transform works.

Here is another way to think about it. Pretend you are on a swing with period three seconds. It swings back and forth every three seconds. The arrows from before are like the kicks on a swing that you time as you try to get higher and higher on the swing. If the kicks are timed off resonance with the swing's natural frequency, so anything other than every three seconds, then you end up slowing down the swing. However, if every kick is timed to match the frequency of the swing, every three seconds, you create resonance, amplifying the swing's motion. If we start with a bunch of states, metaphorically swings, with different periods, than only the swing with the correct period will be moving after a while. It will be the state with the biggest amplitude or highest probability of being observed. Of course, there is no actual dials or arrow paths or swings in a quantum computer. That is just a visual representation of adding complex numbers, which are the amplitudes of waves. Waves and their crazy ability to either reinforce each other with constructive interference, or negate each other with destructive interference, are at the heart of quantum physics. The dial with three dots is showing constructive interference by making the arrow path grow, which represents the likelihood the quantum computer will measure that state. The other dials are destructively interfering, making it less likely we will detect them.

VI. Conclusion

This paper elaborates on some basic concepts applied to quantum computing. It first outlines these key concepts, introduces the mathematics needed for understanding quantum computing and finally explores the Shor's Algorithm as it applies to both classical and quantum computer security.

References Références Referencias

1. Barrett, Paul. "Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor." In Conference on the Theory and Application of Cryptographic Techniques, pp. 311-323. Springer Berlin Heidelberg, 1986.
2. Brown, W. Steven. "On Euclid's algorithm and the computation of polynomial greatest common divisors." Journal of the ACM (JACM) 18, no. 4 (1971): 478-504.
3. Friedman, Jonathan R., Vijay Patel, Wei Chen, S. K. Tolpygo, and James E. Lukens. "Quantum superposition of distinct macroscopic states." nature 406, no. 6791 (2000): 43.
4. Lanyon, B. P., T. J. Weinhold, Nathan K. Langford, M. Barbieri, D. F. V. James, Alexei Gilchrist, and A. G. White. "Experimental demonstration of a compiled version of Shor's algorithm with quantum entanglement." Physical Review Letters 99, no. 25 (2007): 25050.
5. Namias, Victor. "The fractional order Fourier transform and its application to quantum mechanics." IMA Journal of Applied Mathematics 25, no. 3 (1980): 241-265.
6. Wallraff, Andreas, David I. Schuster, Alexandre Blais, and L. Frunzio. "Strong coupling of a single photon to a superconducting qubit using circuit quantum electrodynamics." Nature 431, no. 7005 (2004): 162.

© 2017 Global Journa ls Inc. (US)

Global Journals Inc. (US) Guidelines Handbook 2017

www.GlobalJournals.org