Pace University
DigitalCommons@Pace

Honors College Theses | Pforzheimer Honors College

12-2018

A Com parison of Classical and Quantum Implement ations of a Random Number Generator: An Introduction to Quantum Computing

Charlotte Coffin

Follow this and additional works at: https://digitalcommons.pace.edu/honorscollege_theses

 Part of the Computer Sciences Commons

A Comparison of Classical and Quantum Implementations of a Random Number Generator: An Introduction to Quantum Computing

Student:
Charlotte Coffin
Bachelor of Science in Computer Science

Advisor:
Andreea Cotoranu
Information Technology Department

Presentation date: December 11, 2018
Graduation date: May 2018

Table of Contents

Abstract

Quantum computers have potential to contribute to advancements in many disciplines including chemistry, material science, and artificial intelligence. As quantum computers become more accessible, there is a greater need for people to understand quantum computation to spearhead such advancements. This thesis discusses the basics of quantum computing, including fundamental concepts and a literature review. In order to demonstrate how quantum computation works, a random number generator is implemented on a quantum computer simulator and compared to a random number generator implemented on a classical computer. The results of the two experiments are compared. The results show that a quantum random number generator creates truly random numbers. This work contributes to the basic understanding of quantum computing by computer science novice learners, thus opening quantum computing to further exploration.

Introduction

In 1965, Gordon Moore, co-founder of Intel, predicted that every two years our computing power would double. Titled Moore's Law, this trend has held true for over 50 years [1]. As our computing power increased, our ability to engineer solutions to problems has increased as well. While classical computers are predicted to reach the physical limits of their computation power soon, quantum computing offers an alternative for the future of Moore's Law. However, there is a need to understand the practical applications of quantum computing, and furthermore to introduce novice learners to quantum computation.

This thesis discusses the basics of quantum computing, including fundamental concepts and a literature review. A random number generator is designed, implemented, and tested on both a classical computer and a quantum computer, and the results are compared.

Randomness is an important concept in computation. Random numbers are used extensively in cryptography for example. Random numbers fall into two categories: pseudo-random and truly- random. A pseudo-random number is a number that is generated using a mathematical algorithm or created from a measurement of some part of the universe. Either way, the number follows some pattern and therefore has potential to be predicted, and thus is ultimately not truly random. A truly- random number is a number that cannot be predicted. A quantum computer possesses special qualities that make it possible for truly random numbers to be created [2].

Understanding and improving computation capabilities is a key focus in computer science. In this context, quantum computing has the potential to revolutionize computing as we know it. In order to understand quantum computing, one must understand how a classical

computer operates. A classical computer is binary, such that every operation is done with a bit which can have a value of either 1 or 0. A classical computer implements operations on these bits using gates. All operations in a classical computer are comprised of a combination of these gates [1].

A quantum computer works differently than a classical computer. Instead of using classic bits, a quantum computer uses qubits. The state of a qubit is more complex in that, a qubit can have more than two states, and is often represented as a Bloch Sphere as in Figure 1.

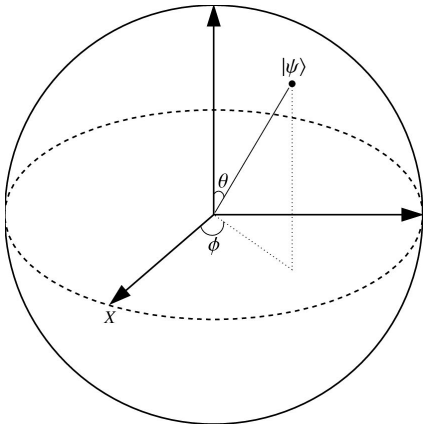

Figure 1: A bloch sphere representing a qubit [15]

Given its property to hold multiple states, a qubit can store much more information than a classical bit. However, extracting meaningful information from a qubit is perhaps one of the biggest challenges in quantum computation.

There are two key properties that make quantum computing different from classical computing: superposition and entanglement. Superposition is a linear combination of states [3]. This means that if a qubit is in superposition, there is an equal chance of these states being measured as 0 as there is of being measured as 1. This can be described effectively with a visual quantum circuit, as described in Figure 2.

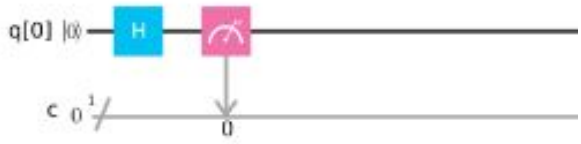

Figure 2: Quantum circuit with one qubit in superposition and one Hadamard gate

In Figure 2, a quantum score is created using a single qubit. This qubit is then put into superposition using a Hadamard gate, and measured 1000 times, as quantum scores often are [4]. The output is in Figure 3 and shows that there is a nearly 50 percent chance of the qubit being in either state 0 or state 1. This result showcases one of the most powerful properties of quantum computing.

Figure 3 Quantum score for one qubit in superposition after 1000 measurements

The other key property of quantum computing is entanglement. Qubits have the ability to become entangled, and this is where much of their computing power comes from. In its most simplistic and abstracted form, entanglement means that while each part of a system is acting randomly, the actions of the entire system are predictable. The behavior of one qubit can be predicted based on the behavior of another qubit. However this property doesn't exist until the

qubits are measured [4]. Each part of a system may act randomly, but when measured, the two parts are strongly correlated. This is explained visually in Figure 4, which shows the quantum score of a system of two qubits put in superposition and then entangled.

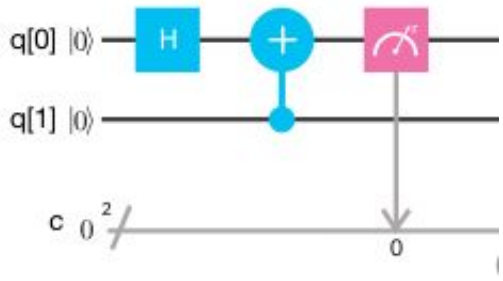

Figure 4: Quantum circuit with two qubits in superposition and entangled

If just one qubit is measured, the output is random, similar to the output of the circuit shown in Figure 2 where half the time the result is zero, and half the time the result is one, as shown in Figure 5.

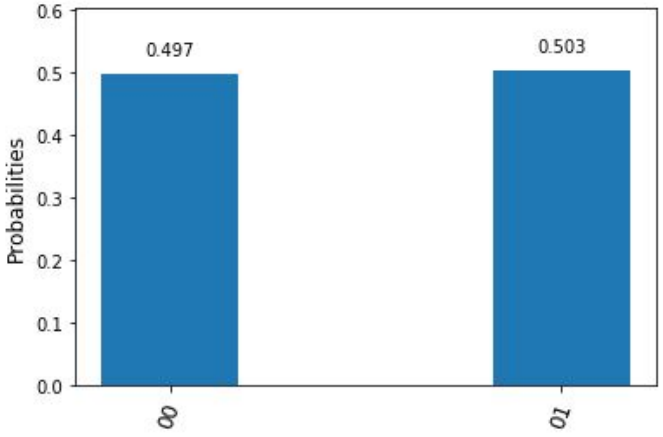

Figure 5: Output for quantum entanglement measuring one qubit

However if both qubits are measured at the same time, the qubits show a strong correlation as shown in Figure 6. This result indicates that while still independently random, both qubits are either 0 or 1, which is due to the qubits being entangled. This property is unique to quantum, and there is no understandable reason for two things that seem totally unrelated to act this way.

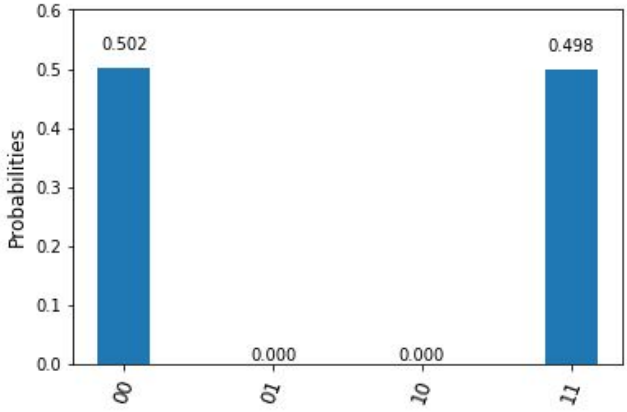

Figure 6: Output for quantum entanglement measuring two qubits

In quantum computing, gates are used to manipulate bits, just as in classic computing. These quantum gates are powerful as they do operations on qubits in superpositioned states, while using significantly fewer bits than a traditional computer would. These quantum gates are the result of matrices multiplied together. A base matrix represents the original state of the qubit, and the gates are set matrices that can be multiplied by the original matrix in order to get a particular output. These quantum gates are strung together into a quantum circuit. All of the operations are calculated and then the state is measured, smashing this information down into a single one-dimensional state; the output is a quantum score [1]. Because of this property, quantum systems can only be measured once, at the end of a program. Additionally, this

measurement will likely be different each time it is taken due to noise in the quantum system. To overcome this limitation, quantum scores are usually measured multiple times, and graphed on a bar plot to indicate the probability of each output [4].

Quantum computing takes advantage of the current understanding that very small particles exist as both a wave and a particle. This means that very small amounts of matter must be measured and understood in order to represent qubits. There are a few different technologies to create a qubit, for example the polarization of a photon, or the spin of an electron [5]. In order for the qubits to work as intended, they must be kept at a very low temperature, near absolute zero, or approximately −459.67 °F [6].

Nowadays, researchers are racing to create the first quantum computer that truly outperforms a current supercomputer. Additionally, they are exploring ways to use the quantum properties to compute, including designing new algorithms suitable for quantum systems.

Literature Review

Quantum computing was created in the last century as a way to understand quantum mechanics. In 1976, Roman Ingarden, a Polish scientist, published a paper on Quantum Information Theory, showing that representing quantum information on a classical computer wouldn't be effective. In 1981, Richard Feynman, a well-known American physicist proposed a quantum computer as the solution to simulating physics on a computer, arguing that classical computers just weren’t sufficient [7]. The theoretical concepts of quantum computing began to take shape in the late 1980s. In 1985, David Deutsch, a British physicist, proposed the first Universal Quantum Computer, and argued that it could exist, and would have properties that

could not be simulated on a classical computer [3]. At this time, coming up with a theoretical quantum computer wasn't impossible, and physicists and mathematicians could develop algorithms that could run on this theoretical machine.

In 2000, David DiVincenzo published a paper outlining the physical requirements of a quantum computer. In his paper DiVincenzo emphasized the physical limitations of quantum computing arguing that it is very difficult to measure and sustain changes to a single electron in the way that is necessary for quantum computation [8]. Later that year, Isaac Chuang created a crude five-qubit quantum computer using Fluoride atoms, and researchers at Los Alamos National Laboratory created a seven-qubit system out of a single drop of water. Since then, more sophisticated machines have been created, but it wasn't until recently that researchers and educators got access to real quantum computers to tackle new problems.

Today, a few companies are racing to build quantum computers. In March 2018, Google unveiled Bristlecone, a 72-qubit quantum chip. Google states that Bristlecone requires more iterations before it can be fully effective, as the amount of system noise increases with the number of qubits, yet they are "cautiously optimistic" about the capabilities of the quantum chip. If this chip will work as intended, then Google will have reached "Quantum Supremacy," defined as the point at which a quantum computer performs an algorithm that a classical computer cannot [9].

IBM, on the other hand, has a few somewhat stable quantum computers that researchers can use. In November 2017, IBM created a 50 - qubit quantum computer, a size which can nearly outperform our current supercomputers. IBM also has a 20 - qubit quantum computer available

for public use. This means that quantum technology is accessible to the general public through the cloud, thus creating opportunities for more research and learning [4].

Another important potential advancement for quantum computing is in the time complexity of certain algorithms. In classical computing, an algorithm acts as a step-by-step instruction manual for a computer to solve a problem. Algorithms are categorized based on their "time complexity," or the rate at which they speed up with respect to the size of the input. Quantum computing has potential to decrease the time complexity of solving difficult problems with the use of new and more efficient algorithms.

Classical algorithms fall into different categories based on the amount of time it takes for them to complete, as follows:

P: a problem that can be solved in polynomial time.

NP: Nondeterministic Polynomial Time, also known as the set of problems where a solution can be verified as correct or incorrect in polynomial time.

NP-complete: a problem where a solution can be verified, but cannot be found in polynomial time [10].

Analysis of a quantum register indicates that one quantum register with *n* qubits can store up to 2^n pieces of information, whereas a classic register can only store *n* pieces of information. This is to say that in order to represent numbers 0 through 128, one would only need log_2 (128 bits), which is the equivalent of 7 qubits. In order to represent that many numbers in classical computers, one would need 128 bits. This it the first hint that quantum could provide a

significant speed up. A classical computer could not represent a quantum one without an exponential slowdown. With this onset of quantum computing offering a potential speed-up, there has been the creation of another class of problems:

QP: a problem that can be solved in polynomial time by a quantum machine.

It wasn't until the *Deutsch–Jozsa algorithm* that this theoretical speed up was shown to exist. This algorithm was one of the first to demonstrate a difference between a classical problem and a quantum problem [11]. While the Deutsch-Jozsa algorithm does not have many direct applications, it is important to note that it is essential to the understanding of the power of quantum computing.

A key example that shows a real life application of quantum computing is *Shor's algorithm*. Originally published in 1994, and expanded upon in 1997 by Peter Shor, the algorithm significantly decreases the speedup of factoring prime numbers. The time complexity of this problem is not yet solved, and no classical algorithm can do it very efficiently [3]. However, Shor's algorithm can find the prime factorization in polynomial time. This speed up indicates that certain types of encryption can now be broken with quantum computation, and because of this capability, we will have to look into alternative ways to encrypt data in the future [10].

Another algorithm that takes advantage of the unique properties of quantum computers is *Grover's algorithm*. In its most simple form, Grover's algorithm manipulates the amplitudes of numbers in a quantum register in order to find a number in a list of numbers. For example, if you

have a thousand database entries, and you are looking for a single entry with a unique attribute, Grover's algorithm could be used. Assuming that no other entry in that list has that attribute, a classical computer would have to go through every entry to check if it had that quality, thus taking linear time. Grover's algorithm can find that entry in O($\sqrt{n}$) time. This speed up is very significant, and has potential to change database searching drastically. Additionally, Grover's algorithm uses something called "amplitude amplification" in order of a specific entry. This amplitude amplification exploits the properties of quantum computation to create the speedup [3].

Currently quantum computation has a few specific applications to science and research. In 2012, researchers found a way to simulate protein folding using a quantum computer [12]. Quantum also has applications in Machine Learning as it can process large amounts of information better than classical computers do. One example of this capability is the aforementioned Grover's algorithm for finding an entry in a database.

Last but not least, researchers also explored generating truly random numbers using quantum computers. A classical computer only has the ability to generate a pseudo-random number. Some algorithms are pure math, utilizing matrices, and large prime numbers to perform calculations that appear random [2]. Another technique for random number generation is using an arbitrary measurement of something in the universe as the basis for computing random numbers. One example is measuring the time between keystrokes on a computer and using that as the basis for the computation. However, the closer to statistically random a number generator is, the more time and space it takes to compute. This makes random number generation a problem in the world of computation.

Research Question

This work aims to observe the differences between a truly random number and a pseudo-random number generator. In order to make such observations, a random number generator is implemented on both a quantum computer and a classical computer. The goal is to see if quantum can create truly random numbers, and to understand some of the differences between quantum computing and classical computing.

Methodology

In order to observe the properties of quantum computation, a random number generator program is written for each computer: classical and quantum. The program for the quantum computer is implemented on QISKit, an open source platform for quantum computing. QISKit is one of the first environments designed to assist with writing quantum programs. QISKit is based on the Python programming language and runs on IBM's quantum computer [13]. The random number generator takes advantage of the superposition property of quantum systems. This property means that until the quantum system is measured, it is both 1 and 0, but once it is measured it is exactly 1 or 0. A screenshot of the code for the random number generator on QISKit is in Figure 8.

Figure 8 shows seven qubits put into superposition. The qubits are in a linear combination of states, i.e. they have an equal chance of being 0 as they do 1 when they are measured. Each of these seven qubits are put into a superposition using Hadamard (H) gates. Mathematically, each qubit is put into $\frac{|0\rangle + |1\rangle}{\sqrt{2}}$, represented as $|+\rangle$. Using seven qubits means that this quantum circuit can represent up to 2^7 states. In this case they are representing

numbers 0 to 127. This program uses 7 of the 20 qubits available on QISKit thus reducing system noise and ensuring that system readings are accurate.

```
qc.h(qr[0]) #Create a superposition on the quantum bit
qc.h(qr[1]) #Create a superposition on the quantum bit
qc.h(qr[2]) #Create a superposition on the quantum bit
qc.h(qr[3]) #Create a superposition on the quantum bit
qc.h(qr[4]) #Create a superposition on the quantum bit
qc.h(qr[5]) #Create a superposition on the quantum bit
qc.h(qr[6]) #Create a superposition on the quantum bit



qc.measure(qr[0], cr[0]) #Measure the bit, and store its value in the clasical bit
qc.measure(qr[1], cr[1]) #Measure the bit, and store its value in the clasical bit
qc.measure(qr[2], cr[2]) #Measure the bit, and store its value in the clasical bit
qc.measure(qr[3], cr[3]) #Measure the bit, and store its value in the clasical bit
qc.measure(qr[4], cr[4]) #Measure the bit, and store its value in the clasical bit
qc.measure(qr[5], cr[5]) #Measure the bit, and store its value in the clasical bit
qc.measure(qr[6], cr[6]) #Measure the bit, and store its value in the clasical bit
```

Figure 8: Sample code: QISKit Quantum random number generator qubit setup

After the quantum circuit is set up, it is executed. Figure 9 is a demonstration of how to run this quantum circuit on IBM's QISKit quantum computer. The results of the quantum score are sent back in a Python Dictionary. Since the quantum circuit is ran only once, the output is a single string of 0's and 1's. However, in order to extract the results, the output is looped through, and stored into a temporary variable. The variable is then converted from a binary number into a decimal number, which is the actual random number.

```
def random():
    job_sim = execute(qc, backend_sim, shots=shots_sim) #Run job on chosen backend for chosen number of shots
    stats_sim = job_sim.result().get_counts() #Retrieve results
    temp = ""
    for x in stats_sim:
        temp += x

    num = int(temp, 2)
    return num
```

Figure 9: Sample code: random number generator function on QISKit

The program for the classical computer utilizes Python's built in random number generator function. The function is based on the *Mersenne Twister algorithm*. The algorithm was introduced in 1997 by Makoto Matsumoto and Takuji Nishimura, and it is based on the Mersenne Prime. The algorithm simulates randomness. However, while the algorithm passes a fair amount of statistical random tests, its output is not truly random [2]. More sophisticated random number algorithms use measurements of the environment, like the time between key presses, to generate a more random number, but those too are not truly random [2].

Results and Discussion

In the classical implementation of this problem, it is impossible to generate a truly random number. The Python random number generator applies mathematical functions to create something that appears random [14]. However a quantum computer is able to create a truly random number, thus making for a better random number generator. This work demonstrates the differences between a basic algorithm implemented on both a quantum computer and a classical computer.

In order to understand the quantum random number generator, 1000 numbers are generated using the quantum system, and then plotted on a bar graph with respect to how many times they appear in the data. The less variation in the amount of times a number appears in the output, the more random the number is, as shown in Figure 10. To compare, 1000 random numbers are generated using the Mersenne Twister (MT) algorithm in Python and the counts are shown in Figure 11.

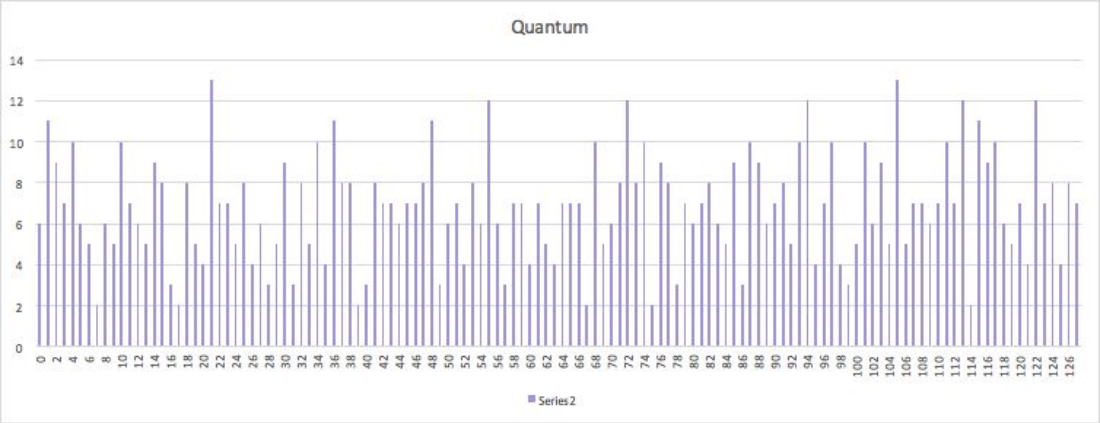

Figure 10: Bar graph of counts of each appearance of a number in 1000 runs of the quantum algorithm

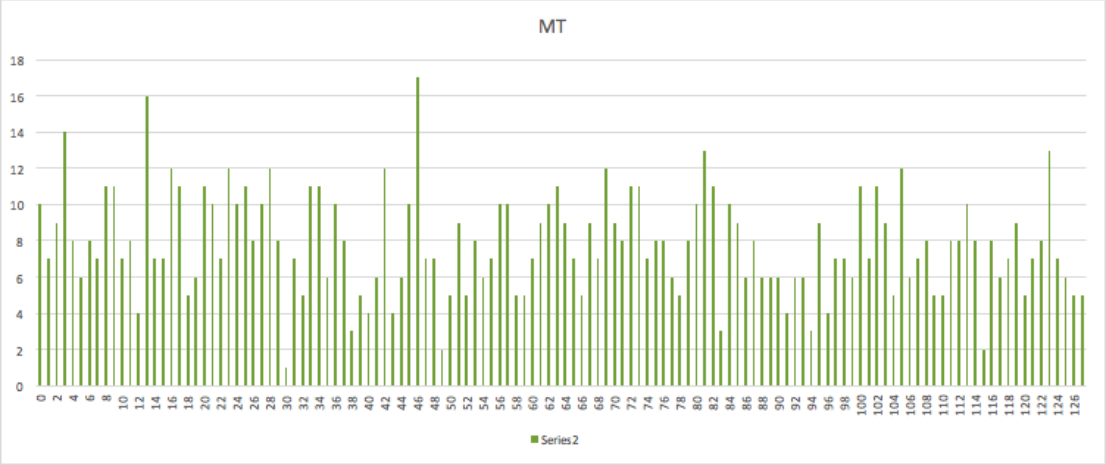

Figure 11: Bar graph of counts of each appearance of a number in 1000 runs of Mersenne Twister

The two graphs look similar with respect to the number distribution. This result is likely related to the Mersenne Twister having an intentional distribution of numbers across its range. However, there are differences between the two as shown in Figure 12.

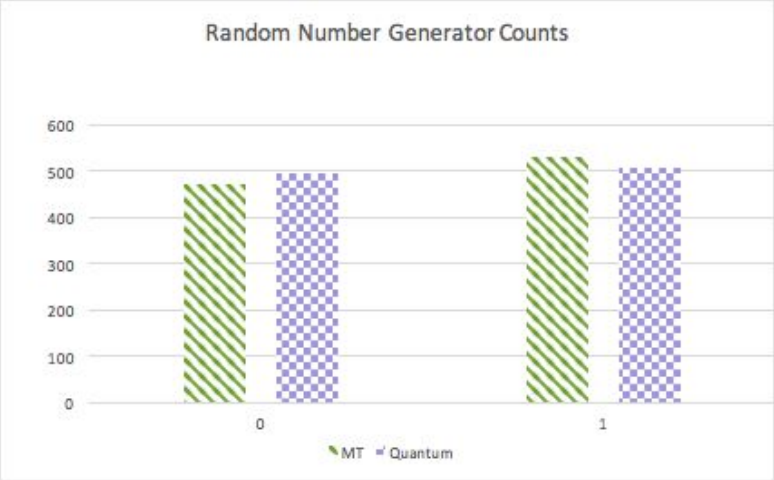

Figure 12: Comparison of Quantum and Mersenne Twister random number generator counts

Figure 12 shows that the output of the quantum algorithm is much closer to being exactly half zero and half one. As indicated in the graph, there is a smaller difference between the two counts in the quantum number generator than there is in the Mersenne Twister algorithm. This shows that the quantum algorithm is more true to the theoretical property that every outcome is equally likely to be in the output. This is how quantum computing generates a truly random number.

There are differences between a quantum random number generator and a pseudo-random number generator implemented on a classical computer, using Python. While there are many creative algorithms to address this problem, it is important to note a truly random system cannot be simulated through mathematical functions. A quantum computer however is an

example of a truly random system, and this property is essential in creating a quantum random number generator.

In addition to this comparison of truly random and pseudo-random numbers, this work is meant to add to the collection of programs that could be completed by a quantum novice. This project covers the fundamental concepts of quantum computing, and aids in the understanding of quantum concepts by those who are trying to learn quantum on their own. This means that quantum computing concepts are more accessible to the general public, and computer science minds can work to solve problems using this technology.

Conclusion

As demonstrated in the design and implementation of a quantum random number generator, there are differences between classical computing and quantum computing. The ability to generate a truly random number has numerous applications in statistics and cryptography. In current cryptographic systems, if a random number can be predicted, the system can be cracked. If cryptographic systems were built on truly random numbers, they would be much more secure. Having truly random numbers has incredible potential in many fields, and is one example of the many uses of quantum computers.

This thesis serves as an introduction into one of the basic uses of quantum computing, and is a vessel to explain the differences between quantum computing and classical computing. A summary of quantum concepts and a review of the literature surrounding this topic were provided. Then, a random number generator was implemented using QISKit and IBM's quantum computer. The random number generator was also implemented using classical computing and a

pseudo-random number generator. The results of these random number generators were analyzed, and the differences were compared.

This work adds to the fundamental understanding of quantum computation, and to the list of programs that have now been implemented on a quantum computer. Quantum computing has potential to revolutionize the way we think about computation, and to impact many fields for years to come. Moore's law may be slowing down, but quantum computing means that Moore's law just might hold true for a couple more years, thus pressing researchers to envision new ways to manipulate information.

References:

[1] N. David Mermin, *Quantum Computer Science: An Introduction*. Cambridge University Press, 2007.

[2] Archana Jagannatam, "Mersenne Twister - A Psuedo Random Number Generator and its Variants" 2010. [Online].

[3] Geoffrey Brumfiel, "D-Wave quantum computer solves protein folding problem," *Nature.com*, 2017. [Online]. Available: http://blogs.nature.com/news/2012/08/d-wave-quantum-computer-solves-protein-folding-problem.html. [Accessed: 20-Nov-2018]

[4] "IBMQ Experience" 2017. [Online]. Available: https://quantumexperience.ng.bluemix.net/qx/tutorial?sectionId=beginners-guide&page=002
-Introduction~2F001-Introduction. [Accessed: Sep. 10, 2018].

[5] E. Conover, "Quantum computers are about to get real," *Science News*, 2017. [Online]. Available: https://www.sciencenews.org/article/quantum-computers-are-about-get-real. [Accessed: 14-Oct-2018]

[6] K. M. Svore and M. Troyer, "The Quantum Future of Computation," *Computer* , vol. 49, no. 9, pp. 21–30, 2016.

[7] R. P. Feynman, "Simulating physics with computers," *Int. J. Theor. Phys.*, vol. 21, no. 6–7, pp. 467–488, 1982.

[8] D. P. DiVincenzo, "The Physical Implementation of Quantum Computation," *Fortschr. Phys.*, vol. 48, no. 9–11, pp. 771–783, 2000.

[9] Google, "A Preview of Bristlecone, Google's New Quantum Processor," *Google AI Blog*. [Online]. Available: http://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html. [Accessed: 14-Oct-2018]

[10] M. Hayward, Quantum computing and Shor's algorithm

[11] D. Deutsch, "Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 400, no. 1818, pp. 97–117, 1985.

[12] Geoffrey Brumfiel, "D-Wave quantum computer solves protein folding problem," *Nature.com*, 2017. [Online]. Available: http://blogs.nature.com/news/2012/08/d-wave-quantum-computer-solves-protein-folding-problem.html. [Accessed: 20-Nov-2018]

[13] "QISKit" 2018. [Online] Available: www.qiskit.com. [Accessed: Sep 08, 2018].

[14] Docs.python.org. (2018). *random — Generate pseudo-random numbers — Python 3.7.2rc1 documentation*. [online] Available at: https://docs.python.org/3/library/random.html [Accessed 19 Dec. 2018].

[15] S. Abramsky, "Contextual Semantics: From Quantum Mechanics to Logic, Databases, Constraints, and Complexity," Contextuality from Quantum Physics to Psychology Advanced Series on Mathematical Psychology, pp. 23–50, 2015.

Appendix

```
#QuantumRandomGenerator.py
from qiskit import *
#7 is 1 - 127
#6 1 -63


IBMQ.load_accounts()
qr = QuantumRegister(7)
cr = ClassicalRegister(7)
qc = QuantumCircuit(qr, cr)


qc.h(qr[0]) #Create a superposition on the quantum bit
qc.h(qr[1]) #Create a superposition on the quantum bit
qc.h(qr[2]) #Create a superposition on the quantum bit
qc.h(qr[3]) #Create a superposition on the quantum bit
qc.h(qr[4]) #Create a superposition on the quantum bit
qc.h(qr[5]) #Create a superposition on the quantum bit
qc.h(qr[6]) #Create a superposition on the quantum bit



qc.measure(qr[0], cr[0]) #Measure the bit, and store its value
in the clasical bit
qc.measure(qr[1], cr[1]) #Measure the bit, and store its value
in the clasical bit
qc.measure(qr[2], cr[2]) #Measure the bit, and store its value
in the clasical bit
qc.measure(qr[3], cr[3]) #Measure the bit, and store its value
in the clasical bit
qc.measure(qr[4], cr[4]) #Measure the bit, and store its value
in the clasical bit
qc.measure(qr[5], cr[5]) #Measure the bit, and store its value
in the clasical bit
qc.measure(qr[6], cr[6]) #Measure the bit, and store its value
in the clasical bit


backend_sim = Aer.get_backend('qasm_simulator')
shots_sim = 1 #Adjust this number as desired, with effects as
described above
```

```
def random():
    job_sim = execute(qc, backend_sim, shots=shots_sim) #Run job
on chosen backend for chosen number of shots
    stats_sim = job_sim.result().get_counts() #Retrieve results
    temp = ""
    for x in stats_sim:
         temp += x

    num = int(temp, 2)
    return num

num = random()
print(num)


#PythonRandom.py
import random;

for x in range(0,1000):
    num = random.randint(0,1)
    print(num)
```