Understanding Quantum Information Systems: Take Your Cue from the Qubit

Andrew Erickson
Dennis Guster
Leena Radeke
Erich Rice
Information Systems Department
St. Cloud State University
720 4th Ave S, St Cloud, MN 56301-4498
dguster@stcloudstate.edu

ABSTRACT

The theory of quantum computing is not exactly new, but considerable progress has been made within the last decade to convert the theory into a physical reality. With many in the field agreeing that this field of computing is sitting on the cusp of its own 'golden age', understanding the true nature and full potential of quantum computing has never been more crucial. The current functionality of a physical quantum computer is neither commonplace nor feasible to maintain within a public setting. Therefore, the authors' purpose is to initiate simulations of quantum algorithms on classical computer using QCL in a LINUX environment. The motive behind this research is to introduce basic elements of quantum computing, in addition to comparing the operational functionality between quantum and classical algorithms when executed on a classical computer. This could benefit IT professionals learning quantum algorithm basics using a commonly configured VM within a cloud.

Key words: quantum computer, qubit, QCL, Shor's algorithm, Grover's algorithm, virtual machine

1 INTRODUCTION

Quantum computing is not exactly a new topic; the concept dates back to the early eighties [6, 9]. Recent advances have made special purpose quantum computers a reality, which has increased interest in the topic [20]. While quantum computing still has a long way to go, it has reached the point where its advantages and disadvantages have been evaluated regarding how it might affect applications running within an enterprise system. Certainly, there are many strengths as well as weakness related to quantum computing [2]; however, it is becoming apparent that the use of quantum computing in some form will grow in years to come. To many, the most important advantage is the increased speed one could expect from a true quantum computer. A quantum computer, of course, will require quantum algorithms. The two most well-known quantum algorithms are Grover's and Shor's [8, 20]. As one would expect, they both can be related to speeding up important computing processes.

Grover's algorithm, when properly implemented on a quantum computer, can perform searches quicker than classical algorithms run on classical computers. To be precise, Grover's algorithm can accomplish an N-sized search in approximately $\sqrt{N}$ time. In mathematical terms, this is a quadratic speed-up over classical methods. The algorithm, if implemented to its full potential, would certainly revolutionize the field of "Big Data".

With Shor's algorithm, the potential speedup may not be welcomed. This is because it is designed to factor an n-bit integer in polynomial time, which would allow for a substantial speed up over classical methods. Therefore, if a quantum computer with enough qubit space became available, it could easily render many widely used encryption methods relying on factoring large numbers obsolete. This is a severe problem, as common encryption algorithms, such as RSA which use keys based on factors, may become vulnerable [16]. Shor's algorithm can provide an exponential speed-up over common classical systems, meaning it could be used to exploit many of the widely used public-key cryptography methods, including RSA.

It appears that quantum computers have become more viable in the past 5 years. However, the field still requires much development before they can be considered as large scale general purpose device. A recent success of a functioning quantum computer is reported by Novet [20]. In part, the problem of developing a true quantum computer can be related to supporting a large number of qubits. A qubit is the way that data is represented in a quantum computer. Qubits are roughly analogous to a bit in a classical device but are also much denser, due to its ability to represent multiple states simultaneously. A major obstacle to the development of a large scale general purpose quantum computer with adequate qubit capacity is the problem of de-coherence [10]. The difficulty stems from the fact that state of the underlying infrastructure (such as a spinning photon) can be easily changed. This makes it difficult to isolate the quantum elements. Furthermore, it is usually the case that the quantum infrastructure supporting the elements requires an ultra-cold environment to function.

Quantum computing may prove to be the infrastructure of the future. Due to its complexity, the time to start learning about this field's advantages and disadvantages, in addition to how it might be deployed, is now. Gaining an understanding of the qubit, which provides a denser way to represent data, is a good place to start. This density is provided in part by the concept of superposition. The principle of superposition allows a qubit to be in multiple states simultaneously. In comparison to a classical bit, a qubit can be in state 0, 1, or both simultaneously. A single 2-bit register can store only one of four possible values at a time: 00, 01, 10, 11; conversely, a 2-qubit register is able to store all four of these values at once. This ability results in the additional density. In terms of processing power, a quantum computer can process 2^N computations, where N is the number of qubits, making it the ultimate parallel processing machine.

To gain a better understanding of a qubit and how they would work in a quantum computer, it is helpful to evaluate their use in a quantum computing simulation language. There are opportunities to secure time on is on a quantum computer, for the simple task of evaluating the basics of qubits. However, it makes more sense to use a quantum computing simulation language for this purpose. One of the most popular of these languages is QCL (Quantum Computing Language). This language is attractive due to its similarities to C, and its compatibility within a LINUX environment makes it an excellent research and instructional tool.

Consequently, the purpose of this paper will be to introduce qubits using the QCL/LINUX environment. Additionally, the fundamentals of quantum computing, such as superposition and gates, will also be discussed on an operational level. Ultimately, the goal is to delineate in detail how a qubit differs from a classical bit, while illustrating how qubits increase data density and provide a foundation to make quantum computing exponentially quicker than classical computing.

2 REVIEW OF LITERATURE

2.1A Brief History of Quantum Computing

The history of quantum computing can be traced back to the early 1970s, although it did not pick up traction until the 1980s with the work of Paul Benioff [1], Richard Feynman [9], and Yuri Manin [15]. The work of David Deutsch [6], which suggested the concept of a universal (or general purpose) quantum computer, was a huge step forward. This concept would be analogous to the universal Turing machine in classical computing. Deutsch's model expected that a universal quantum computer would have the ability to simulate any other quantum computer [8].

There was a lag in findings until Peter Shor developed an integer factorizing algorithm, which could be used to break some classical encryption techniques. This resulted in increased awareness and interest in quantum computing [22]. Specifically, Shor's

algorithm operates more quickly than classical methods due to its ability to solve complex factors in in polynomial time [7]. Theoretically, this algorithm is capable of cracking public-key cryptography in mere minutes if run on a quantum computer with adequate qubit space [3]. These encryption algorithms include many popular schemes, with RSA being the most notable.

Following the development of Shor's algorithm, Lov Grover devised a quantum database search algorithm [11]. This algorithm complements classical algorithms in need of search capability by adding Grover's algorithm as a subroutine. In the early 2000s, advances in the underlying architecture began appearing. For example, the concept of linear optical quantum computing was realized with the implementation of the KLM protocol [13]. This system used linear optical elements, photon sources, and photon detectors. Since then, there have been many advances within the field of quantum computing; one of the most prominent being the development of a special purpose quantum computer by the D-wave company [20].

2.2 Qubits the Fundamental Building Block

The building block that represents data in classical computing is the bit, which can only be in either states 0 or 1. In quantum computing, data is not only represented by qubits 0 or 1, but also by both simultaneously through the concept of superposition. Qubits can be represented by a variety of physical architectures such as ions, atom, photons or electrons. Just like a classical computer, quantum devices require control devices that act likes processors to coordinate operation in memory. When storing qubits, it is often said they are stored in a quantum register. Since a quantum computer uses qubits, which can contain these multiple states simultaneously, its potential is huge. In fact, if it has adequate qubit capacity, a quantum computer could easily be millions of times than the most powerful supercomputer currently available.

To put this into perspective mathematically, the ability to store 0 and 1 simultaneously facilitates the quantum equivalent of "parallel computing". The potential for this technology is massive, as the degree of parallelism doubles with each additional qubit. Mathematically, this can be described as the computational power scales exponentially with the addition of each qubit. Note that the number of states possible interacts with the number of qubit used, such that a 50-qubit system can perform a parallel calculation within a system size of 2^{50}. This value is approximately the size of the memory of current supercomputers. At 300 qubits, the equivalent parallel computing size 2^{300} is approximately equivalent to the number of atoms in the universe, which would never be possible on a classical computer architecture [17].

One of the major obstacles to achieving a quantum computer with a large qubit capacity is the problem of de-coherence. Interference from the environment and other qubits can inadvertently alter the state of a qubit, making it difficult to maintain the states imposed by the "superposition" of quantum states. Solving this de-coherence problem has major benefits not only for quantum computing, but for quantum communication as well.

Currently, there is ground-breaking work underway that would significantly reduce the de-coherence problem. Historically, the underlying infrastructure has been based on quantum atoms trapped in magnetic fields. However, maintaining an atom within a vacuum is problematic, therefore the idea of trapping them in solids begs investigation.

In some recent work at MITT, researchers created a qubit design employing nitrogen atoms embedded in a synthetic diamond structure. This architecture is based on the following axiom: "When nitrogen atoms happen to be situated next to gaps in the diamond's crystal lattice, they produce "nitrogen vacancies," which enable researchers to optically control the magnetic orientation, or "spin," of individual electrons and atomic nuclei. Spin can be up, down, or a superposition of the two" [12]. This system uses a transparent solid, which allows light to pass in both directions, and may not be possible with other solids. More specifically, a crystal structure is like glass: its atoms are regular, and its electronic structure is well defined. The researchers' choice of a diamond crystal is well suited for capturing an atom. This is because the nuclei of diamonds are relatively free of magnetic dipoles, which often result in noise that can cause de-coherence.

2.3 State of Quantum Computers

Currently, the development of a true general purpose physical quantum computer development remains in a preliminary stage, and the promise of a large-scale quantum computer continues to be classified as theoretical technology of the future. Nevertheless, the topic has gained much attention, and research from both a theoretical and practical perspective is on-going. A good example of this is the IBM Quantum Experience (QX) project. This project allows anyone with an internet connection to access IBM'S quantum processor via its supporting cloud architecture. This has created an incredible opportunity for anyone connected to the IBM Cloud to conduct their own experiments and simulations on a real quantum computer. A quantum composer GUI is provided to facilitate usage; alternatively, the quantum computer can also be programmed using quantum assembly language code.

Additionally, quantum computers are now available for commercial use, with D-Wave Systems at the forefront. The D-Wave One is only capable of running discrete optimization math problems. The D-Wave Two followed shortly after, containing a total of 512 qubits. While this is an impressive accomplishment, not all the D-Wave Two's qubits are useable due to de-coherence. A benchmark test in 2013 revealed that the D-Wave could solve two mathematical problems consisting of 100+ variables within half a second [18]. This feat showed that the D-Wave Two was about 3,600 times faster than the CPLEX, which performed the same test in about half an hour. Subsequently, the D-Wave 2X was released about two years later with an impressive 1,000 qubits set within a Chimera graph architecture. The logic behind the 2X was to enact hardware changes aimed at improving performance, rather than just focusing on speed alone.

Google is also involved in quantum computing, and recently has set a goal of producing a 49-qubit chip by the end of 2017. Currently, two successful dry runs have taken place, on

9x1 and 2x3 arrays of qubits. IBM has also made clear that it intends to be a player in the quantum computing field, with the goal of developing a 50-qubit chip within the next five years to support their QX project. Fifty qubits seem to be the magic number, as a quantum computer chip with 50+ qubits would be powerful enough to surpass classical computers on selected tasks [5].

2.4 Overview of Quantum Programming Languages

Quantum languages running on classical computers are available, and can be quite useful from an experimental, simulation, or educational perspective. In fact, one QCL will be used to illustrate qubit usage in detail within this paper. While quantum computing languages don't function like classical computing languages, they are very useful as a means of comprehending quantum algorithms and their application to quantum computing by expressing those algorithms with high-level constructs [19]. There are a number of these languages presently available, which can be classified into two main groups: imperative and functional. In the experiments that follow, the imperative quantum programming language QCL will be used. QCL was an excellent choice for this topic because its syntax and data types are comparable to C programming language. This is a logical background many programmers have, and QCL is designed to work in much the same manner as any language within a LINUX environment. It is thought that this familiarity will reduce the learning curve for experienced programmers when making the switch to the quantum programming world.

3 METHODOLOGY AND RESULTS

The Quantum Computing Language (QCL) in a LINUX environment will be used to illustrate how qubits are used in quantum computing. As stated earlier, QCL is a simulated quantum computing language replicating quantum computing on a classical device. Three examples were developed and tested using QCL. The first example deals with the measurement of qubits in superposition and how it effects their state. The goal of this example is to illustrate that a qubit can be in states beyond just the traditional 0 and 1, as seen in classical bits. The second example is designed to introduce the concept of entanglement, illustrating simple entanglement using two quantum registers. The last example was devised to provide a practical case of how quantum concept might be applied by using entanglement to obtain random results. One of the advantages of the quantum world is the ability to produce truly random results, and quantum random number generators are recognized for their efficiency. The code used herein provides an overview of how logic gates in conjunction with entanglement could be used to implement random results.

3.1 Example 1: Superposition of Qubits

The figures below provide three examples of the effect of measuring a qubit after being transformed into a superposition state by a Hadamard gate. A single qubit register is used in the first example, and the register size is incremented by one in each subsequent example. A Hadamard gate is a building block in quantum computing, and is one of many options within a model that uses quantum circuits to facilitate computation. Typically, logic circuits such as the Hadamard gate perform their operations on a small number of qubits. In the example below, the value is from 1 to 3 qubits. When the Hadamard gate is applied to a single qubit, it will map the basis state to |0> and |1> such that there is an equal probability of being in either state. This means that the qubit could contain 0, 1 or both at the same time. This is done by rotating the physical media, such as a single photon, by a certain degree. Here is an over simplification of the process: in the 0 state, one could picture a sphere rotated to 90° straight up would be 0° and represent the 1 state. If the Hadamard gate is applied, the photon is rotated to 45°, which is half way between 0 and 1 states. Hence, there is a 50/50 chance of the qubit being in either state.

```
BCRL\dennis.guster@eros:/qcl-0.6.4$ ./qcl --dump-format=b
QCL Quantum Computation Language (64 qubits, seed 1505939473)
[0/64] 1 |0>

qcl> qureg q[1];
qcl> reset;
[1/64] 1 |0>
qcl> H(q);
[1/64] 0.70711 |0> + 0.70711 |1>
qcl> measure q;
[1/64] 1 |0>
qcl> dump q;
: SPECTRUM q: <0>
1 |0>
```

Figure 1: Single Qubit Register

As expected, the quantum register "q" has an initial value of 100% chance of being 0 when evaluating the code in the first example. The register should initialize to that state, but the reset command is run just to be sure, which forces the register into that state. Next, the Hadamard gate is applied (H). After going through the Hadamard gate, the superposition states are 50% for 0 and 1. Note that because vector notation is used, the value is reported as a square root. $0.70711^2 = 50$, therefore giving the 50% probability. After being measured, "q" reverts to its original state of 100% chance of being 0. This is because any measurement on a qubit alters its state. However, the simulation language has a dump command that allows one to see the value without changing the state, which is quite useful for educational purposes. Moreover, the fact that measuring a qubit changes its state is just as useful in data communication, as one can tell if the transmitted data has been read by an eavesdropper. Nevertheless, one qubit is not enough for most complex analysis, so the example is replicated using first two and then three qubits. These examples provide interesting and somewhat different results.

In the second instance of the first example, which has the register size defined as two qubits, there are four possible states after the Hadamard gate is applied with a 25% ($.5^2$) chance for each state. In evaluating how this is represented on the physical level using the simplified method utilized above, one must realize with two qubits the address space in increases. Instead of the finite state 0 and 1, there are now four states (2^2). These states are

00, 01, 10, 11. After the application of the Hadamard gate on the 2-qubit register, the possible rotations also increase to four vectors. The span from 0° to 90° is 90°, which is divided by 4 to equal 22.5°. Therefore, a vector in between 0 and 22.5 degrees represents a 25% chance of the value being 00, with a 75% of it not being 00. It then follows that a vector in between 22.5 and 45 degrees would indicate a 25% chance of the state 01 and a 75% chance of it not being 01, and so on. As before, the state becomes altered once it has been measured, in this case to 100% chance of being 11.

```
qcl> qureg q[2];
qcl> reset;
[2/64] 1 |00>
qcl> H(q);
[2/64] 0.5 |00> + 0.5 |01> + 0.5 |10> + 0.5 |11>
qcl> measure q;
[2/64] 1 |11>
qcl> dump q;
: SPECTRUM q: <0,1>
1 |11>
```

Figure 2: Two-Qubit Register

A similar scenario occurs in example three, where the register size is increased to three qubits. As expected, the superposition probabilities are 12.5% ($.35355^2$) for the eight combinations of the 3-qubit field, but the values are modified to a 100% chance of the state 000 when measured. Using the simplified example to support the eight combinations of 0s and 1s, the vector boundaries would need to occur every 11.25°, providing eight zones between 0° and 90°.

```
qcl> qureg q[3];
qcl> reset;
[3/64] 1 |000>
qcl> H(q);
[3/64] 0.35355 |000> + 0.35355 |001> + 0.35355 |010> + 0.35355 |011> +
0.35355 |100> + 0.35355 |101> + 0.35355 |110> + 0.35355 |111>
qcl> measure q;
[3/64] 1 |000>
qcl> dump q;
: SPECTRUM q: <0,1,2>
1 |000>
```

Figure 3: Three-Qubit Register

3.2 Example 2: Simple Entanglement Using Two Quantum Registers

As important as the concept of superposition is regarding the increased density of quantum computing, the concept of entanglement advances it further. Entanglement can be viewed as a nonlocal property enabling qubits to achieve a higher correlation of state relationship when compared to classical systems. A good example of this is the BB84 experiment, which depicts secure quantum communication. In this experiment, two entangled quantum bits are separated and transferred to Alice and Bob. When Alice measures her qubit, she has an equal probability of getting either |0 or |1. Because the qubits have been entangled, the second qubit will act as the first, and Bob will get the same results even if he is 100 kilometers away. In this way, the concept of nonlocality is confirmed.

```
BCRL\dennis.guster@eros:/qcl-0.6.4$ ./qcl --dump-format=b
QCL Quantum Computation Language (64 qubits, seed 1505939986)
[0/64] 1 |0>
qcl> qureg q[1];
qcl> qureg r[1];
qcl> reset;
[2/64] 1 |0,0>
qcl> H(q);
[2/64] 0.70711 |0,0> + 0.70711 |1,0>
qcl> CNot(r,q);
[2/64] 0.70711 |0,0> + 0.70711 |1,1>
qcl> measure q;
[2/64] 1 |0,0>
qcl> dump;
: STATE: 2 / 64 qubits allocated, 62 / 64 qubits free
1 |00>

qcl> qureg q[2];
qcl> qureg r[2];
qcl> reset;
[4/64] 1 |00,00>
qcl> H(q);
[4/64] 0.5 |00,00> + 0.5 |01,00> + 0.5 |10,00> + 0.5 |11,00>
qcl> CNot(r,q);
[4/64] 0.5 |00,00> + 0.5 |01,00> + 0.5 |10,00> + 0.5 |11,11>
qcl> measure q;
[4/64] 1 |11,11>
qcl> dump;
: STATE: 4 / 64 qubits allocated, 60 / 64 qubits free
1 |1111>
qcl> dump q;
: SPECTRUM q: <0,1>
1 |11>
qcl> dump r;
: SPECTRUM r: <2,3>
1 |11>
```

Figure 4: Entangled Qubits Stored in Registers "q" and "r"

The goal of this example is to illustrate how two qubits can be become entangled, with one qubit stored in register "q" and the other qubit in register "r". Once again, we go back to the state interrelationships as the register size increases. Two instances are provided below. The first instance features two registers, each with a size of one qubit. The Hadamard gate is applied, which puts register "q" in a superposition state. The CNot() gate will flip q if r = 1. This should put the two qubits in an entangled state. Note that after the CNot gate is applied, "r" had a value of 1. "q" returns a value of 0 once it has been measured, and by some sort of spooky means so does "r". This is because "q" and "r" have been entangled. Can entanglement work with more than just single qubits? The second instance addresses this question. Once again, the CNot gate flips the "r" register to 1's. When "q" is read as 11, the value in the "r" matches with the value 11.

3.3 Example 3: Using Two Quantum Registers to Generate Random Results

The code below uses two quantum registers to generate random results in the form of the words displayed. The purpose of each command is described by a comment within the code. As one would expect, there are four combinations of results when two registers are

used, which are displayed in the output below. Within the source code highlighted in yellow are the qubit values. Note they are reversed, and leading zeroes are not displayed. The combinations follow with the qubit values in the 3rd column:

00 fabulous turtles	0
01 fabulous porcupines	10
10 magical turtles	1
01 magical porcupines	11.

```
BCRL\dennis.guster@eros:/qcl-0.6.4/lib$ cat EntangleEx.qcl
procedure EntangleEx() {
int valueA;
int valueB;       //values we use later after measuring entangled
qubits

qureg a[1];
qureg b[1];       //quantum registers we use to store data

Not(a);           //set value in qureg a[1] to 1
H(a);             //set value in qureg a[1] to superposition (same as a
                  // Mix()) of 50% 0 and 50% 1
CNot(b,a);        //entangle qubit a and b, "copy" value of a into b...
                  //and perform the Not operation

dump a;
dump b;           //dump showing this is true

measure a,valueA; //measure a into valueA collapsing superposition.

print "-------------";
dump a;           //prints a, showing it was measured and collapsed
dump b;           //prints b, showing that measuring a also collapsed b
                  //and that they were entangled!

measure b, valueB; //store value of b to be used later

      if valueA == 0{
            print "Fabulous";
      } else {
            print "Magical";
      } if valueB == 0{
            print "Turtles";
      } else {
            print "Porcupines";
      }
            //prints random word combination depending on results of
            //entanglement/superposition collapse.
            //run the program a few times and you will get different
            //combinations depending on how the qubits collapsed!
}
```

Figure 5: Two Quantum Registers Generating Random Results in the Form of Words Displayed

By looking at the output, we can see superposition probabilities within each register are 50% either 0 or 1 in each case, so any value in the table above has an equal opportunity to appear. But wait a minute…we are running a quantum algorithm through simulation on a classical computer—how random can that be? There are numerous cautions about the lack of true randomness within classical computers. An excellent summary is provided by NIST (National Institute of Standards). The code is then run below and depicts results that contain all four possible outcomes. To ascertain how random the results would be in this classical

environment, one would need to run the code many times. If run on a true quantum computer, true random results can obviously be expected.

```
BCRL\dennis.guster@eros:/qcl-0.6.4$ ./qcl --dump-format=b
QCL Quantum Computation Language (64 qubits, seed 1506533768)
[0/64] 1 |0>
qcl> include "EntangleEx.qcl";
qcl> EntangleEx();
: SPECTRUM a: <0>
0.5 |0>, 0.5 |1>
: SPECTRUM b: <1>
0.5 |0>, 0.5 |1>
: -------------
: SPECTRUM a: <0>
1 |1>
: SPECTRUM b: <1>
1 |1>
: Magical
: Porcupines
[0/64] -1 |11>
qcl> EntangleEx();
: SPECTRUM a: <0>
0.5 |0>, 0.5 |1>
: SPECTRUM b: <1>
0.5 |0>, 0.5 |1>
: -------------
: SPECTRUM a: <0>
1 |0>
: SPECTRUM b: <1>
1 |1>
: Fabulous
: Porcupines
[0/64] -1 |10>
qcl> EntangleEx();
: SPECTRUM a: <0>
0.5 |0>, 0.5 |1>
: SPECTRUM b: <1>
0.5 |0>, 0.5 |1>
: -------------
: SPECTRUM a: <0>
1 |1>
: SPECTRUM b: <1>
1 |0>
: Magical
: Turtles
[0/64] 1 |1>
qcl> qcl> EntangleEx();
: SPECTRUM a: <0>
0.5 |0>, 0.5 |1>
: SPECTRUM b: <1>
0.5 |0>, 0.5 |1>
: -------------
: SPECTRUM a: <0>
1 |0>
: SPECTRUM b: <1>
1 |0>
: Fabulous
: Turtles
[0/64] 1 |0>
```

Figure 6: EntangleEx.qcl Output

4 DISCUSSION AND CONCLUSIONS

Although the results obtained from running the QCL code were simplistic in quantum computing terms, they did illustrate how a qubit could be defined and manipulated. More interesting problems could certainly have been run on a true quantum computer, but the goal of these experiments was simply to provide a basic foundation of the qubit structure. For this introductory educational exercise, the classical computer was adequate. Two of the most important treatments of qubits superposition and entanglement were illustrated. To keep the topic simple, a limited number of qubits were used in each case. Hopefully, the reader was able to connect these basic examples to the super-density possibilities of qubits in which the computation scaling power is based on 2^N, where N is the number of qubits employed.

The fact that QCL is a C-like language, interacting as such with the operating system, makes its use relatively easy for an experienced programmer. Perhaps the biggest change is the use of logic gates. This paper dealt with the Hadamard and CNot gates, two of the most common gates. The Hadamard gate was used to put qubit(s) into a superposition state where it could be both a 0 and 1 simultaneously, which was done by rotating the underlying physical structure. In the example provided, a 0 was 90° and 1 was 0°; therefore, the 50/50 state was situated at 45°. The CNot gate was used to entangle multiple qubits, ensuring the entangled qubits would return the same value if only one was measured.

Many people are dreading or denying the advent of quantum computing. However, based on the numerous commercial products currently available that are related to quantum computing and quantum data communication, it appears that the importance of this technology can only continue to grow. Because of its added complexity, the learning curve for this technology is much steeper than it is for classical technology. Additionally, there is not a direct correlation to classical technology, and so a limited amount of classical information technology directly transfers to the quantum world. Because quantum computing technology appears to be the wave of the future, the time to start gaining knowledge of the quantum world is now. This paper presented a basic foundation of the manner data is represented and stored in quantum world: the qubit. As previously stated, one of the primary advantages of quantum programming on classical computers is that it provides a sandbox to learn the concepts, and to get a basic understanding of how qubits might be deployed and manipulated. The authors found it especially useful to run trials to see how the size of the qubit registers influence the results and effected scaling. Furthermore, the code helps illustrate how qubits behave and demonstrates how a single qubit can be represented by a vector containing a whole column of numbers (the probabilities of superposition), as opposed to just two discrete bit values.

The rationale for this paper came about through the authors' desire to express the importance of spending a little time with a quantum computing language, which will provide a useful foundation in understanding the technology, with the first recommended topic of study being the concept of a qubit. Based on the observations herein, at least for the short term, it appears that quantum computing cannot reach its full potential until a true general-purpose quantum computer is made available. However, progress continues to be made. Therefore, proactive procedures are needed to prepare future IT professionals for

the quantum computing revolution, and how this development may change the landscape in the field of Information Technology.

5 REFERENCES

[1] Benioff, P. (1982). Quantum mechanical Hamiltonian models of Turing machines. *Journal of Statistical Physics, 29*(3), 515-546.

[2] Bennett, C. H., Bernstein, E., Brassard, G., & Vazirani, U. (1997). Strengths and weaknesses of quantum computing. *SIAM journal on Computing, 26*(5), 1510-1523.

[3] Bernstein, D. J. (2010, May). Grover vs. McEliece. In *International Workshop on Post-Quantum Cryptography* (pp. 73-80). Springer, Berlin, Heidelberg.

[4] Byrne, C. (2015, May). The Golden Age of Quantum Computing is Upon Us (Once We Solve These Tiny Problems). Retrieved from https://www.fastcompany.com/3045708/big-tiny-problems-for-quantum-computing.

[5] Courtland, R. (2017, May 24). Google aims for quantum computing supremacy [News]. *IEEE Spectrum,54*(6), 9-10. doi:10.1109/mspec.2017.7934217

[6] Deutsch, D. (1985, July). Quantum theory, the Church-Turing principle and the universal quantum computer. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* (Vol. 400, No. 1818, pp. 97-117). The Royal Society.

[7] Deutsch, D., & Jozsa, R. (1992, December). Rapid solution of problems by quantum computation. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* (Vol. 439, No. 1907, pp. 553-558). The Royal Society.

[8] De Wolf, R. (2013). Quantum Computing: Lecture Notes. Retrieved November 2017, from http://homepages.cwi.nl/~rdewolf/qcnotes.pdf.

[9] Feynman, R. P. (1982). Simulating physics with computers. *International journal of theoretical physics, 21*(6), 467-488.

[10] Gottesman, D. (2009, April). An introduction to quantum error correction and fault-tolerant quantum computation. In *Quantum information science and its contributions to mathematics, Proceedings of Symposia in Applied Mathematics* (Vol. 68, pp. 13-58).

[11] Grover, L. K. (1996, July). A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing* (pp. 212-219). ACM.

[12] Hardesty, L. (2015). Qubits with staying power. http://news.mit.edu/2015/extended-qubits-quantum-computers-0129.

[13] Knill, E., Laflamme, R., & Milburn, G. J. (2001). A scheme for efficient quantum computation with linear optics. *Nature*. Nature Publishing Group. 409(6816), 46-52.

[14] Lenstra, H. W., & Pomerance, C. (1992). A rigorous time bound for factoring integers. *Journal of the American Mathematical Society, 5*(3), 483-516.

[15] Manin, Y. (1980). Vychislimoe i nevychislimoe (computable and noncomputable). *Soviet Radio* (pp. 13–15). In Russian.
[16] Marchenkova, A. (2015, August 15). Break RSA encryption with this one weird trick. Retrieved November 2017, from https://medium.com/quantum-bits/break-rsa-encryption-with-this-one-weird-trick-d955e3394870.
[17] Martinis, J. (2017, 16 May). People of ACM - John Martinis. https://www.acm.org/articles/people-of-acm/2017/john-martinis.
[18] McGeoch, C. C., & Wang, C. (2013, May). Experimental evaluation of an adiabiatic quantum system for combinatorial optimization. In *Proceedings of the ACM International Conference on Computing Frontiers* (p. 23). ACM.
[19] Miszczak, J.A. (2012). High-level structures for quantum computing. *Synthesis Lectures on Quantum Computing, 4*(1), 1-129.
[20] Novet, J. (2015, December 2). Google says its quantum computer is more than 100 million times faster than a regular computer chip. Retrieved November 2017, from http://venturebeat. com/2015/12/08/google-says-its-quantum-computer-is-more-than- 100-million-times-faster-than-a-regular-computer-chip/.
[21] Rahmi, M., Shamanta, D., & Tasnim, A. (2012). Basic Quantum Algorithms and Applications. *International Journal of Computer Applications, 56*(4).
[22] Shor, P. W. (1994, November). Algorithms for quantum computation: Discrete logarithms and factoring. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on* (pp. 124-134). IEEE.
[23] Understanding the Hadamard gate on the Bloch sphere. (2017, February 22). Retrieved from https://physics.stackexchange.com/questions/313959/understanding-the-hadamard-gate-on-the-bloch-sphere/314600