Teaching Quantum Computing

Lewis Westfall and Avery Leider
Seidenberg School of CSIS, Pace University, Pleasantville, New York

***Abstract*—There is little research or information on how best to train students on the important ideas of quantum computing. This is a startling observation, as quantum computing is a revolution in information processing whose discoveries are being made at a rapid pace. Hundreds of millions of dollars are being invested in quantum computers by nation-states and by companies. To innovate, develop, operate and maintain these quantum computers takes a well educated workforce that understands what is happening and what the objectives are. In an effort to achieve with quantum computing the point at which the quantum computer is more efficient than a classical computer at some tasks, intense efforts are underway in multiple global locations, where, for example, microwave engineers are added to the quantum computing team, or software engineers, programmers, heating and cooling engineers or computer engineers. Where are they getting their quick and intense training to introduce them to the team? On site, presumably. Who is training and teaching them? Is it expected of the hard pressed researchers, never trained in the best way to teach, themselves, to be taken away from their priority time to familiarize the newcomers? We are two PhD Computer Science candidates, experiencing the learning of quantum computing for the first time, and we wish to share our experiences and best discoveries to help others shorten their time to understanding. It is our belief that the best time to introduce quantum computing now is in a honors level high school course, or as an undergraduate. A high school student approaches quantum computing from an unbiased perspective, with an attitude that is positive about their future and the role of quantum computing in it. We present that it is possible to teach high school students in such a way that the mathematics and physics of quantum computing feels more natural to learn than classical computing.**

Keywords*—*Quantum Information Systems, Quantum Computing, Pedagogy

I. Introduction

There is very little research or information on how best to train students on the important ideas of quantum computing. Challenged by our Quantum Computing course for PhD students of Computer Science, we searched extensively for better materials than our heavily cited textbook[1] and our professors' lecture slides [2] and our search turned up dry.

Physics is making quantum computing possible, however the driver for the massive investment in the field is computer science, with hundreds of millions of dollars being invested in quantum computers by nation-states and by companies.

Quantum computing development has been ongoing for several decades. Something recently has changed. The sudden demand for qualified quantum computing personnel is because of the recent leap forward in technology with successful quantum computers in 2016 when IBM put a small quantum computer on the cloud. According to one of the authors of our primary textbook[1] in this course, Isaac Chuang, MIT Professor, "...quantum computing is actually real. It is no longer a physicist's dream – it is an engineer's nightmare.[3]"

When Google in 2018 announced its latest 72-qubit quantum computer[4], named Bristlecone, it said that Bristlecone will achieve Quantum supremacy. This quantum supremacy is the prize that all quantum computing efforts are reaching for - it is the point at which the Quantum computer is better at computations than the best classical computer (in the mind of Google, that means supercomputers) can be [5]. Quantum supremacy is not universal - there will always be some tasks that classical computers will do better than quantum computers. The IBM and Google quantum computers are built in similar ways. Microsoft, DWave, Intel and IonQ have quantum computer ventures, with different designs.

Teaching information technology and computer science subjects is done in a different manner than the teaching of physics. We recommend using flashcards[6], that use a time sensitive algorithm to enhance memory formation. Flashcards are helpful for memorizing essential details that will speed up comprehension of quantum computing, similar to how a person learning computer programming must memorize some commands, methods and steps in order to get to the business of learning programming.

Additionally, we propose a free short interactive online course, that can be reached by either mobile phone or computer.

Our project is to teach quantum computing and show by example how it can be taught better than it is now.

In this study, we present the key fundamental ideas to learn in order to program quantum computers.

II. Literature Review

Kaye, Laflamme and Mosca's[7] introduction to quantum computing quickly covers the basics, and then launches with ever increasing complexity into the quantum algorithms that are the centerpiece of the book, with a crescendo reached in the quantum error correction chapter. The problems of error correction are vital to the eventual success of quantum computing, so this book is right to focus on them.

Kitaev, Yu and Vyalyi's[8] book on classical and quantum computing starts as its foundation for Part 1 the logic of classical computing, ending on probability and games in the classical sense, and then in Part 2 draws the reader into quantum computing, drawing similarities and differences to illustrate. In Part 3, the authors have written out 57 pages containing all solutions to every problem in the book, with careful detail that completely explains the answers.

Marinescu and Marinescu[9] approach quantum computing as a mathematical abstraction, which most quantum computing

works do, but with the pessimistic idea that useful quantum computers are decades away. The authors start their textbook discussion in physics, and discuss light waves and photons at length in the first chapter as contrasted to Kitaev, et al, who start their text with classical computing. Interestingly, the Marinescus only need tensor product and vectors throughout the book to explain all of the quantum ideas. The Walsh-Hadamard Transform, Fast Hadamard Transform, Modular Arithmetic, Abstract Algebra, Chinese Remainder Theorem, and Fourier Transform are separated out as math that is too difficult, and these subjects are all put in the appendix.

Nielsen and Chuang[1] have written the most fundamental book on the subject of Quantum Computing. It has, at this writing, according to Google Scholar, been cited by other researchers 30,762 times in less than 10 years. However, it has many flaws. The text is peppered with engaging exercises and problems, but no solutions. It makes several assumptions in the mathematical formulas, assumptions that would be natural to a mathematician or a physicist, but not natural to a computer scientist. These assumptions make the steps hard to follow. It explains some ideas long after the reader needed them to understand a concept. It could be that it is cited so much because it was one of the first textbooks published on the topic of quantum computing, and not because it is terribly good.

Reiffel and Polak[10], in their gentle introduction, place their emphasis on definitions, and making definitions between different but similar ideas more clear. They also feel it necessary to explain the entanglement of multiple qubits, this confusing area, that is beyond the purposeful entanglement of a pair of qubits. Their book is theoretical, yet they take pains to explain decoherence, fault tolerance, and error correction - all of which are part of the actual quantum computing environment.

III. Project Requirements

Classical computing uses bits, the smallest piece of data, of 1 or 0, off or on. All of its logic gates are built on this principle. The qubit or quantum bit, the smallest piece of data in quantum computing, can be 0 or 1 or any value in between and not just on a one dimensional line but in 3 dimensional space.

The Bloch Sphere is used to visualize a single qubit. The North pole is the zero ket, the South pole is the one ket, and the other vectors on the sphere are representations of the superpositions.

The value of a qubit is represented by the equation

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

$|0\rangle$ and $|1\rangle$ are examples of a physics notation, called kets, and they represent vectors. α and β represent complex numbers which fall inclusively between 0 and 1. The values of α and β are limited because they must satisfy the equation $\alpha^2 + \beta^2 = 1$. α^2 is the probability that the value of the qubit is 0 and β^2 is the probability that the value is 1. The initial value of of the qubit is $|0\rangle$, in Equation 1, where the $\alpha = 1$ and the $\beta = 0$.

The operators on the qubits are called gates and are analogous to the logic gates (and, or, and not) in classical computing. These gates are represented as matrices. A gate that operates on a single qubit is represented by a 2x2 matrix. Gates that operate on more the one qubit are correspondingly larger.

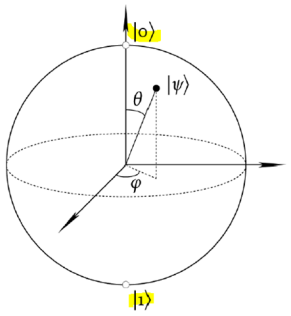

Figure 3.2: The Bloch sphere representation of a qubit. Any qubit $|\psi\rangle$ admits a representation in terms of two angles θ and φ where $0 \leq \theta \leq \pi$ and $0 \leq \varphi < 2\pi$. The state of any qubit in terms of these angles is $|\psi\rangle = \cos(\theta/2)\,|0\rangle + e^{i\varphi}\sin(\theta/2)\,|1\rangle$.

Fig. 1. Bloch Sphere [11]

A. Linear Algebra

Because the qubit is represented by a vector and the operations on the qubit or multiple qubits is done by matrix manipulation, some knowledge of linear algebra is required. Don't worry, the amount you need is only a small part of linear algebra and also the easier part.
The essentials[12]

Scalar is a number which can be complex, eg. a + bi

Vector A vector has a magnitude and a direction. A column vector is a n x 1 matrix, while a row vector is a 1 x n matrix. Each element of a vector is a scalar and represents a dimension.

- Vector is a mathematical concept.
- Ket - $|\varphi\rangle$ - is a physics concept and is a vector in Hilbert space and is a column vector.
- Bra - $\langle\varphi|$ - is the conjugate transpose of a ket and is a row vector.

$$|\varphi\rangle = \begin{bmatrix} a \\ b \end{bmatrix} \tag{2}$$

$$\langle\varphi| = \begin{bmatrix} a & b \end{bmatrix} \tag{3}$$

Matrix A matrix is n x m and every element is a scalar. Below is a 3 x 4 matrix.

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \end{bmatrix} \tag{4}$$

Vector and Matrix Addition To do vector or matrix addition, simply add the corresponding element.

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} a_0 + b_0 \\ a_1 + b_1 \\ a_2 + b_2 \end{bmatrix} \tag{5}$$

$$\begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} + \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix} = \begin{bmatrix} a_{00}+b_{00} & a_{01}+b_{01} \\ a_{10}+b_{10} & a_{11}+b_{11} \end{bmatrix} \quad (6)$$

Multiplication by a scalar To multiply by a scalar, simply multiply each element by the scalar.

$$s \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} sv_1 \\ sv_2 \end{bmatrix} \quad (7)$$

$$s \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} = \begin{bmatrix} sa_1 & sb_1 \\ sa_2 & sb_2 \end{bmatrix} \quad (8)$$

Transpose The superscript T identifies the transpose operation. The transpose is done by swapping the top right element with the bottom left element and similarly swapping each element on one side of the diagonal with the corresponding element on the other side of the diagonal. The diagonal itself is not changed.

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix}^T = \begin{bmatrix} a_{00} & a_{10} & a_{20} \\ a_{01} & a_{11} & a_{21} \\ a_{02} & a_{12} & a_{22} \end{bmatrix} \quad (9)$$

Taking the transpose of a column vector gives a row vector and the transpose of a row vector is a column vector.

$$\begin{bmatrix} a \\ b \end{bmatrix}^T = \begin{bmatrix} a & b \end{bmatrix} \quad (10)$$

$$\begin{bmatrix} a & b \end{bmatrix}^T = \begin{bmatrix} a \\ b \end{bmatrix} \quad (11)$$

Conjugate The superscript * identifies the conjugate operation. The conjugate is taken by changing the sign of the imaginary part of each element. Minus (-) to plus (+) and plus to minus. If the vector or matrix has no imaginary part, then there is no change.

$$\begin{bmatrix} a+bi & c-di \\ m+ni & s+ti \end{bmatrix}^* = \begin{bmatrix} a-bi & c+di \\ m-ni & s-ti \end{bmatrix} \quad (12)$$

Conjugate Transpose The superscript dagger (†) indicates the conjugate transpose operation. First the conjugate of the vector or matrix is taken and then the transpose.

$$\begin{bmatrix} a+bi & c-di \\ m+ni & s+ti \end{bmatrix}^\dagger = \begin{bmatrix} a-bi & m-ni \\ c+di & s-ti \end{bmatrix} \quad (13)$$

Product of a vector and a matrix The result of this operation is vector with the size and shape of the original vector. The size of the vector can be 1xn or nx1 and the size of the matrix must be nxn.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} ae+bf \\ ce+df \end{bmatrix} \quad (14)$$

$$\begin{bmatrix} a & b \end{bmatrix} \begin{bmatrix} e & g \\ f & h \end{bmatrix} = \begin{bmatrix} ae+bf & ag+bh \end{bmatrix} \quad (15)$$

Inner product The inner product of two vectors is a scalar. The inner product of two matrices of size n x m and m x p is a matrix of size n x p. Both are represented by $\langle A \mid B \rangle$.

$$\begin{bmatrix} a & b \end{bmatrix} \begin{bmatrix} c \\ d \end{bmatrix} = ac + bd \quad (16)$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & g \\ f & h \end{bmatrix} = \begin{bmatrix} ae+bf & ag+bh \\ ce+df & cg+dh \end{bmatrix} \quad (17)$$

Tensor product

$$|A\rangle \otimes |B\rangle = |A\rangle\,|B\rangle = |AB\rangle \quad (18)$$

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, B = \begin{bmatrix} e & g \\ f & h \end{bmatrix} \quad (19)$$

$$\begin{aligned} |AB\rangle &= \begin{bmatrix} a & b \\ c & d \end{bmatrix} \otimes \begin{bmatrix} e & g \\ f & h \end{bmatrix} \\ &= \begin{bmatrix} a\begin{bmatrix} e & g \\ f & h \end{bmatrix} & b\begin{bmatrix} e & g \\ f & h \end{bmatrix} \\ c\begin{bmatrix} e & g \\ f & h \end{bmatrix} & d\begin{bmatrix} e & g \\ f & h \end{bmatrix} \end{bmatrix} \\ &= \begin{bmatrix} ae & ag & be & bg \\ af & ah & bf & bh \\ ce & cg & de & dg \\ cf & ch & df & dg \end{bmatrix} \end{aligned} \quad (20)$$

Standard Gates

$$Identity = \boxed{\text{I}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (21)$$

$$Pauli - X = \boxed{\text{X}} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad (22)$$

$$Pauli - Y = \boxed{\text{Y}} = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \quad (23)$$

$$Pauli - Z = \boxed{\text{Z}} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad (24)$$

$$Hadamard = \boxed{\text{H}} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad (25)$$

That's it. This is all the linear algebra you need for quantum computing.

B. Other Algebra

There are powerful algorithms that one would want to know in order to do higher level quantum computing programming to achieve cryptography goals, factor prime numbers, perform search or sort.

However, these are at the theoretical level, because the quantum computers available today are not capable of executing them. This challenge will not be solved in the near future, because of the necessity of staging the qubits in preparation to calculating on them with these powerful algorithms. Qubit readiness is perishable, and if it takes too long to get them all set up, their readiness expires. However, these algorithms offer great promise for the future, especially the Quantum Fourier Transform.

1) Quantum Fourier Transform: The Quantum Fourier Transform, proposed first in 1998[13], is a building block for several other quantum algorithms. Quantum Fourier Transform is actually a Discrete Fourier Transform modified for Quantum. It can be written as a formula:

The discrete Fourier transform acts on a quantum state (note the x)

$$\sum_{i=0}^{N-1} x_i \left|i\right\rangle \tag{26}$$

and maps it to a quantum state (note the y):

$$\sum_{i=0}^{N-1} y \left|i\right\rangle \tag{27}$$

However the simplest way to use the Quantum Fourier Transform in quantum computing is as a matrix. One takes the complex vector needing to be understood and multiplies it by the Quantum Fourier Transform matrix, and the result is a simplified vector that is more useful.

$$F_N = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \omega_n^3 & \cdots & \omega_n^{N-1} \\ 1 & \omega_n^2 & \omega_n^4 & \omega_n^6 & \cdots & \omega_n^{2(N-1)} \\ 1 & \omega_n^3 & \omega_n^6 & \omega_n^9 & \cdots & \omega_n^{3(N-1)} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega_n^{N-1} & \omega_n^{2(N-1)} & \omega_n^{3(N-1)} & \cdots & \omega_n^{(N-1)(N-1)} \end{bmatrix}.$$

Fig. 2. Fourier Transform Matrix [14]

IV. Representing qubits:

The ket $\left|0\right\rangle$ is the vector

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

and the ket $\left|1\right\rangle$ is the vector

$$\begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

.

A system of two qubits has the base values $\left|00\right\rangle, \left|01\right\rangle, \left|10\right\rangle$, and $\left|11\right\rangle$. Similarly to equation 1, the full description of a two qubit state is

$$\alpha_{00} \left|00\right\rangle + \alpha_{01} \left|01\right\rangle + \alpha_{10} \left|10\right\rangle + \alpha_{11} \left|11\right\rangle. \tag{28}$$

The values of $\alpha_{00}, \alpha_{01}, \alpha_{10}, \alpha_{11}$ a11 must satisfy the equation

$$\alpha_{00}^2 + \alpha_{01}^2 + \alpha_{10}^2 + \alpha_{11}^2 = 1. \tag{29}$$

The vector representation of a two qubit ket is the tensor product of the two individual qubits. e.g.

$$\left|00\right\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{30}$$

An important two qubit stat is the Bell State, also called the EPR Pair.

V. Gate Operations

The initial state of a qubit is $\left|0\right\rangle$. This represents the vector

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

. Applying the Identity gate (I) to $\left|0\right\rangle$ gives

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \left|0\right\rangle \tag{31}$$

Applying the Pauli - X gate (X) gives

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \left|1\right\rangle \tag{32}$$

Applying the Pauli-Y gate (Y) gives

$$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ i \end{bmatrix} = i \left|1\right\rangle \tag{33}$$

Applying the Pauli-Z gate (Z) gives

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \left|0\right\rangle \tag{34}$$

Applying the Hadamard gate (H) gives

$$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \left|0\right\rangle + \frac{1}{\sqrt{2}} \left|1\right\rangle \tag{35}$$

After applying the X gate to $\left|0\right\rangle$ give $\left|1\right\rangle$ in equation 32. Applying the gates to $\left|1\right\rangle$ gives the following results

$$I \left|1\right\rangle = \left|1\right\rangle \tag{36}$$

$$X \left|1\right\rangle = \left|0\right\rangle \tag{37}$$

$$Y \left|1\right\rangle = -i \left|1\right\rangle \tag{38}$$

$$Z \left|1\right\rangle = - \left|1\right\rangle \tag{39}$$

$$H \left|1\right\rangle = \frac{1}{\sqrt{2}} \left|0\right\rangle - \frac{1}{\sqrt{2}} \left|1\right\rangle \tag{40}$$

It is important to note here that all of the gate operations except the Hadamard gate result in the value being either 0 or 1 with 100% probability if the original qubit is in a state that has 100% probability. If the original qubit is in a superposition state then it remains in a superposition state. All these functions can be done with classical logic operations.

Now it starts to get interesting. The Hadamard gate introduces superposition to the qubit. This means that it has a value of 0 with the probability of $\left|\alpha\right|^2$ and 1 with the probability of $\left|\beta\right|^2$. If the Hadamard gate operates on a pure state, i.e. $\left|0\right\rangle$ or $\left|1\right\rangle$, the values of α and β are $\frac{1}{\sqrt{2}}$ and the probabilities are $\left(\frac{1}{\sqrt{2}}\right)^2$ or $\frac{1}{2}$.

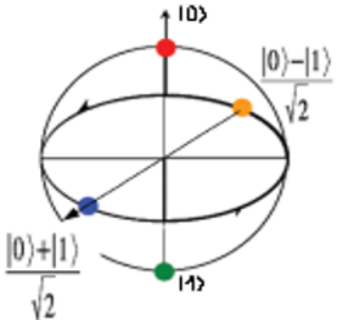

Fig. 3. Bloch Sphere showing basis states and applied Hadamard [15]

VI. Multiple Qubit Gates

[12] In general a multiple qubit gate is an operation where one qubit controls the gate on another qubit. The value of the control qubit determines if the the action of the target qubit's gate is active or not. If the value of control qubit is 0 then nothing happens to the target qubit. However if the value of the control qubit is 1 then the target qubit's gate operates on the target qubit.

The first of these gates that we will look at is the control not or CNOT gate. The target qubit has an X (not) gate that is controlled by another qubit. The depiction of the CNOT gate is shown in Figure 4.

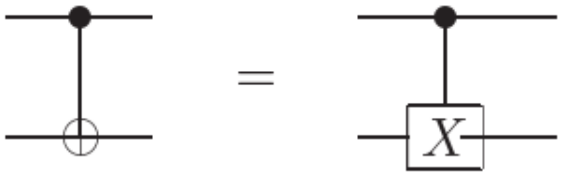

Fig. 4. Two representations of the CNot Gate [1]

Let the control qubit be a and the target qubit b

$$a = \alpha\,|0\rangle + \beta\,|1\rangle = \alpha \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \beta \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \tag{41}$$

$$b = \gamma\,|0\rangle + \delta\,|1\rangle = \gamma \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \delta \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \gamma \\ \delta \end{bmatrix} \tag{42}$$

If the control qubit is 0, basis state $|0\rangle$, then the target qubit is not affected, but if the control qubit is 1, basis state $|1\rangle$, then the target qubit is flipped.

This is all straight forward, but if the control qubit started as a 0 and has been operated on by a Hadamard gate, then its state will be

$$\frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle \tag{43}$$

. This says that the control bit will be 0 half the time and 1 the other half and that the target bit will be flipped half the time. Now both qubits are in superposition and they are entangled. Entanglement is beyond the scope of this paper.

Superposition and entanglement are what separate quantum computing from classical computing where a bit can have a value of only 0 or 1. These features allow the quantum system to be in multiple states at the same time and are only resolved when the state is measured. The catch is that the quantum program must be run many times, usually starting at a minimum of 1024 executions, and the results tabulated to get the solution.

VII. Measurement

There is one other operation that is important. It is the measurement operation. When a measurement is taken of a qubit, it will return 0 or 1 and the state of the qubit is lost. It can no longer be used for quantum processing.

Since each qubit can have a probabilistic value and the measurement can only return 0 or 1, one execution of the program will not give a reliable result. The probability that it will return 0 is determined by α and 1 by β. One run of the quantum program will give a probabilistic answer. To get around this problem, the program is run multiple times. The result is the plot of how many times each result occurred. 1024 executions is considered the fewest executions to give a realistic answer but several thousand executions may be required for programs that have complex processes.

VIII. Examples

The simplest way to try a quantum computer program is to use the free service by IBM Q Experience. For example, the Pauli X gate is known as a bit flip gate, because if you start with a $|0\rangle$ and apply the Pauli X gate, and then measurement gate, you will receive a $|1\rangle$ as a result.

Figure 5 shows what that quantum computing program (five Pauli X gates on five $|0\rangle$s then measurement) looks like in the Composer, a graphic user interface:

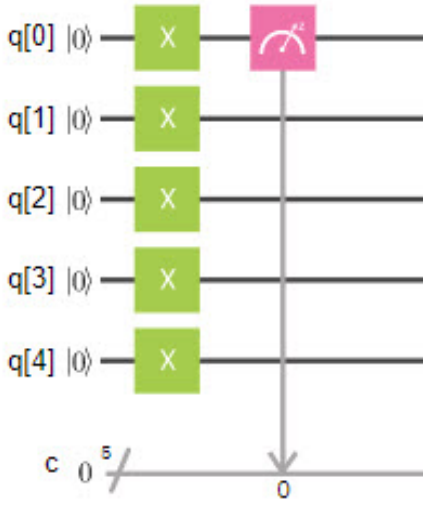

Fig. 5. Pauli X flips the bits in Composer

In Figure 6 we see what that same quantum computing program (Pauli X gate then measurement) looks like in the Quantum Assembly Language:

```
qreg q[5];
creg c[5];

x q[0];
x q[1];
x q[2];
x q[3];
x q[4];
measure q[0] -> c[0];
```

Fig. 6. Pauli X flips the bits in QASM

What that QASM code is saying is: prepare a quantum register of 5 qubits, and a classical register of 5 qubits. Then do a Pauli X gate operation on each qubit. After the Pauli X, measure only qubit 0, and put the results in the classical register. The result of our program is 00001.

Note that the qubits are labeled 0 through 4, instead of 1 through 5, and that the 0 qubit is on top and the high order qubit is on the bottom. When the results are displayed the 0 qubit is on the right and the high order qubits are to the left in ascending order.

A. *Basis States[16]*

The first example we will look at is the basis states for a qubit, which are $|0\rangle$ for the ground state and $|1\rangle$ for the excited state. When the qubit is created it is in the ground state and can be put in the excited state by passing the qubit through an Pauli-X (X) gate.

The first section of code creates a quantum register (qr) and a classical register (cr). Each with size 1.

```
# Creating registers
qr = Q_program.create_quantum_register('qr', 1)
cr = Q_program.create_classical_register('cr', 1)
```

The next section creates the quantum circuit (ground) using the quantum register qr and the classical register cr. It then measure qr and places the result in cr.

```
# Quantum circuit ground
qc_ground = Q_program.create_circuit('ground', [qr], [cr])
qc_ground.measure(qr[0], cr[0])
```

This section creates another quantum circuit (excited) using qr and cr. After qr has been created, it is passed through an X gate and then measured with the result being placed in cr.

```
# Quantum circuit excited
qc_excited = Q_program.create_circuit('excited', [qr], [cr])
qc_excited.x(qr)
qc_excited.measure(qr[0], cr[0])
```

The two circuits, ground and excited, are translated from Python into QASM, the Quantum ASseMbly language.

```
circuits = ['ground', 'excited']

Q_program.get_qasms(circuits)
```

The resulting QASM code represents two programs that create and measure the state of the qubit. The first creates and measures the ground state and the second creates a ground state qubit, converts it into the excited state using an X gate, and then measurers it.

```
OPENQASM 2.0;
include "qelib1.inc";
qreg qr[1];
creg cr[1];
measure qr[0] -> cr[0];

OPENQASM 2.0;
include "qelib1.inc";
qreg qr[1];
creg cr[1];
x qr[0];
measure qr[0] -> cr[0];
```

The execute command runs the program runs the program

```
result = Q_program.execute(circuits,
backend='ibmqx_hpc_qasm_simulator', shots=1024,
max_credits=3, wait=10, timeout=240)
```

The histograms are created using the commands

```
plot_histogram(result.get_counts('ground'))
plot_histogram(result.get_counts('excited'))
```

The ground state histogram Figure 7 shows that the value 0 has a probability of 100% and the excited state histogram Figure 8 has 100% probability of being 1.

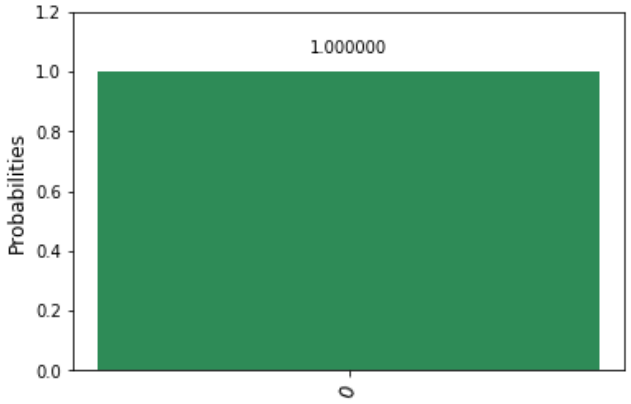

Fig. 7. Ground State Histogram

B. *Superposition[16]*

To explore superposition we create the quantum register, qr, and the classical register, cr, both with size 1, as we did in basis state example,

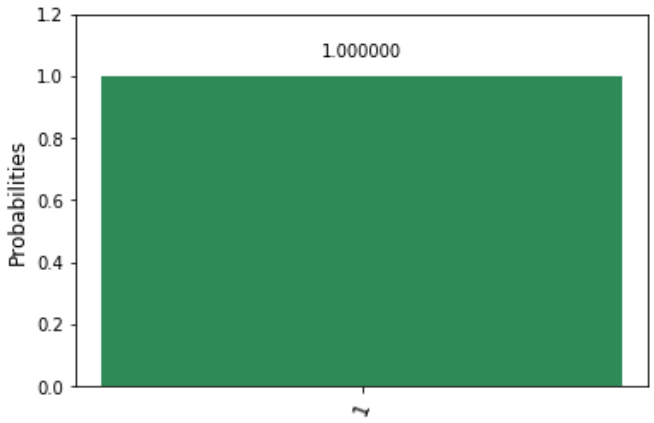

Fig. 8. Excited State Histogram

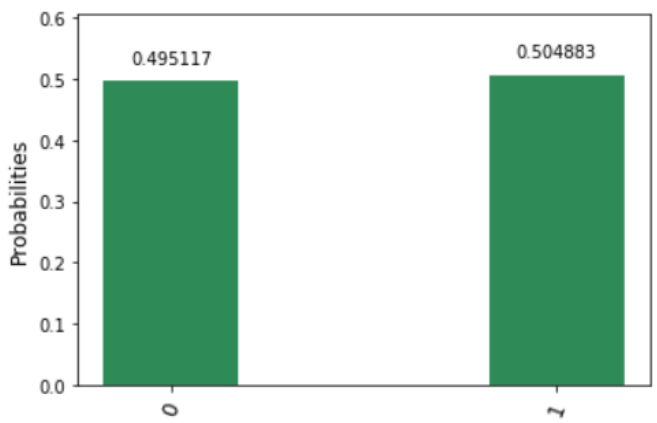

Fig. 9. Entangled State Histogram run on the simulator

```
# Creating registers
qr = Q_program.create_quantum_register('qr', 1)
cr = Q_program.create_classical_register('cr', 1)
```

Then we create the quantum circuit superposition using the registers qr and cr with the qubit originally being in the basis state $|0\rangle$. We impose superposition on the qubit by passing it through a Hadamard (H) gate. This puts the qubit in the superposition state, $\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$. Finally we measure the result.

```
# Quantum circuit superposition
qc_superposition = Q_program.create_circuit('superposition',
[qr], [cr])
qc_superposition.h(qr)
qc_superposition.measure(qr[0], cr[0])
```

The program is run and the histogram created.

```
result = Q_program.execute(circuits, backend=backend,
shots=shots, max_credits=3, wait=10, timeout=240)

#plot_histogram(result.get_counts('superposition'))
```

The resulting QASM code creates the quantum and classical registers, applies the H gate to the qubit, and measures the qubit.

```
OPENQASM 2.0;
include "qelib1.inc";
qreg qr[1];
creg cr[1];
h qr[0];
measure qr[0] -> cr[0];
```

The histogram Figure 9 shows that the results are approximately 50/50 between 0 and 1. Note that this was run on the simulator. When the program is run a real quantum computer, the values for all four possible states, equation 28, are listed. Figure 10

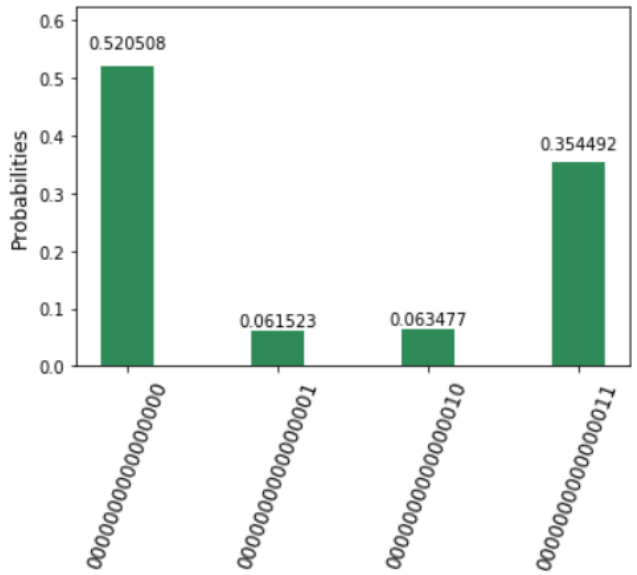

Fig. 10. Entangled State Histogram run on a quantum computer

IX. Methodology

Research by instructors of a massively open online course (MOOC) recently has shown the advantages of using the learning concept of chunking when teaching math and science.[17]. Chunking assumes that when the learning mind is confronted with more information that it knows how to process, it does not learn quickly. So Chunking is an approach to solve this bottleneck. The student masters a small bit of information that is closest to them, conceptually. The student memorizes bits of information. Each small bit of information, each small chunk, that is mastered or memorized or both, frees up that space in the mind that it formerly occupied when it was a mystery. That open space is now free for the student to make associations, creative insights, and leap forward with new conclusions and thought that were not possible before when they were overwhelmed with the subject.

We propose chunking as an approach to how the information to be learned in Quantum Computing is delivered to the students, in two ways: using the flashcard system to

remember key concepts, and in using an online course, that we have started working with, on Udemy. An online course, free, makes it easy for a student to revisit material until they are confident. Also, we mean to use as contributors other students who are learning quantum computing for the first time, so that we capture their fresh insights that are a part of learning an engaging and challenging topic such as this one, and incorporate them into the online course.

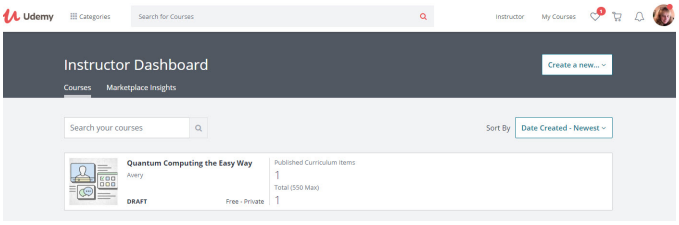

Fig. 11. Udemy Course

X. Conclusion and Future Work

In this paper we have identified the most fundamental ideas about quantum computing that are necessary to understand it, and we have pruned a lot of unnecessary information. The reason for the large amount of highly technical information in quantum computing textbooks is that they were written for and by mathematicians and physicists. As such they expound, at length, on the implementation of quantum circuits. This information is important for developing quantum computers, but is not necessary for writing quantum applications. Most references of the subject of quantum computing were published when the topic was mostly theoretical. It has only very recently become practical and of use to computer scientists.

We have set out here all the necessary knowledge that needs to be understood in order for a high school student or college undergraduate to start writing and running simple quantum computer programs. We are building a draft course with supporting draft flashcards to use for teaching the ideas of quantum computing to that audience that we have found to be essential. Our future work is to complete our teaching course and flashcards, and then use them. We will test our students' knowledge before and after our program of instruction, and observe them, to see if they can write quantum programs with confidence as a result of our instruction.

Acknowledgment

The authors would like to thank Prof Ronald Frank, PhD, Prof Charles Tappert, PhD, and Istvan Barabasi.

References

[1] M. A. Nielsen and I. Chuang, "Quantum computation and quantum information," 2002.

[2] C. Tappert, *Lecture Slides from Quantum Computing Course at Pace University*. Available at http://csis.pace.edu/ctappert/cs837-18spring/index.htm.

[3] W. Knight, "MIT technology review," *Serious quantum computers are finally here. What are we going to do with them?*, 2018.

[4] A. Li, *Google Quantum AI Lab Bristlecone processor*. Available at https://9to5google.com/2018/03/05/google-quantum-ai-lab-bristlecone-processor/.

[5] R. F. Mandelbaum, *Google Unveils Largest Quantum Computer Yet*, 2018. http://www.gizmodo.co.uk/tag/quantum-computing/.

[6] D. Elmes, *Anki flashcards*. Available at https://apps.ankiweb.net.

[7] P. Kaye, R. Laflamme, and M. Mosca, *An introduction to quantum computing*. Oxford University Press, 2007.

[8] A. Y. Kitaev, A. Shen, and M. N. Vyalyi, *Classical and quantum computation*. No. 47, American Mathematical Soc., 2002.

[9] D. C. Marinescu and G. M. Marinescu, *Approaching quantum computing*. Pearson/Prentice Hall, 2005.

[10] E. G. Rieffel and W. H. Polak, *Quantum computing: A gentle introduction*. MIT Press, 2011.

[11] M. M. Wilde, "From classical to quantum shannon theory," *arXiv preprint arXiv:1106.1445*, 2011.

[12] Ronald Frank PhD, *Linear Algebra Review*. Class notes.

[13] J. Preskill, "Lecture notes for physics 229: Quantum information and computation," *California Institute of Technology*, vol. 16, 1998.

[14] *Quantum_Fourier_transform Processing 2018*. Available at https://en.wikipedia.org/wiki/Quantum_Fourier_transform.

[15] R. Van Meter , *Introduction to Quantum Computing*. Available at http://www.soi.wide.ad.jp/class/20050012/slides/01/index_44.html.

[16] S. Barabasi, *Midterm Assignment 2: Superposition and Entanglement*. Jupyter Notebook including Python code.

[17] E. D. Lovell and D. Elakovich, "Developmental math pilot: Massive open online course (mooc), psychology concepts, and group work," *Community College Journal of Research and Practice*, vol. 42, no. 2, pp. 146–149, 2018.