An Analysis of Quantum Error Correction Methods and Entangled Logical Qubits

by David Shimkus, Bachelor of Science

A Thesis Submitted in Partial
Fulfillment of the Requirements
for the Degree of
Master of Science
in the field of Computer Science

Advisory Committee:

Dr. Thoshitha Gamage, Chair

Dr. Mark McKenney

Dr. Eren Gultepe

Graduate School
Southern Illinois University Edwardsville
December, 2023

© Copyright by David Shimkus December, 2023
All rights reserved

ABSTRACT

AN ANALYSIS OF QUANTUM ERROR CORRECTION METHODS AND ENTANGLED LOGICAL QUBITS

by

DAVID SHIMKUS

Chairperson: Professor Thoshitha Gamage

Presented here are findings on implementing entangled logical qubits. This work was motivated by the need to better understand the quantum physics behind these computations, make the topic more approachable, and improve space efficiency of quantum error correction algorithms. The goal of the work is to illustrate that the explored methods can help detect and correct errors in quantum computers, and to set reasonable expectations in this regard. Towards that direction, this work provides examples of four different quantum error correction methods running on three different platforms. Included here is a novel method of implementing the Steane quantum error correction code using less auxiliary qubits than the traditional method. Engineering challenges related to transitioning these algorithms from simulations to physical quantum computers is then discussed and groundwork is laid for further experiments.

ACKNOWLEDGEMENTS

I would like to thank my thesis committee, friends, family, and coworkers for their support. I am also thankful for the excellent textbook "Quantum Computing for Computer Scientists" by Noson Yanofsky and Mirco Mannucci of which has the best introductory material on this subject I have found. To Christine I send my love.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Different problems call for different solutions. Computer scientists are interested in finding efficient solutions to problems by optimizing time or space resources. This might include changing an algorithm to arrive at the same output for some given input in a fewer amount of computational steps, or by using a smaller memory footprint. Invoking a wide variety of parallelization concepts can provide further opportunities for optimization on specific hardware types. Eventually all classical methods of optimization for certain problems may be exhausted or closed (sometimes by proof). Does this mean that scientists are to stop looking for optimization opportunities? By no means! Argued below is that by looking at Quantum Computing methods one can arrive at algorithmic speedups and savings that are not feasible with classical computing methods. Quantum computers are certainly not a silver bullet to all problems, but when applied thoughtfully and with an understanding of the underlying principles one can extrapolate new solutions to certain problems.

These concepts are not always apparent or intuitive to classically trained scientists or programmers. As such, simulating quantum computing with classical computers can be a useful learning tool. The nature of these simulations require large and dense types of linear algebra computations. This lends itself well to using reduced instruction set computing (RISC) architectures such as graphics processing units (GPU's). These systems are more readily available than physical quantum computers which further lends credence to their usefulness in this regard.

Simulating quantum circuits with GPU platforms can demonstrate logical qubit entanglement and help provide benchmarks, insights, and understanding into algorithms and methodologies used by extant and future physical quantum systems. But what is a "logical"

qubit? How are qubits entangled? What is the purpose of entanglement? Chapter 2 dives into a high-level explanation of these concepts along with the quantum physics principles upon which the theory depends. This is followed by Chapter 3 which looks into how these qubits are utilized by quantum computers to perform algorithms and introduces the complexity analysis. Chapter 4 discusses at length Quantum Error Correction (QEC) techniques from the current literature.

What is the tradeoff for implementing error correction techniques? Hypothesized here is that implementing certain QEC techniques provides a positive benefit in resisting different types of errors with associated probabilities. Also demonstrated is that even though these QEC techniques are proven mathematically, there are engineering limitations to current physical quantum computers in their realization of these mathematics.

This material is intended to be understood as a general introduction to this topic, with more granular details presented for specific use cases to help communicate the ideas presented. Readers with a Computer Science, Mathematics, or Physics backgrounds will get more or less out of different sections, but may all benefit from the experimental results and code modules that have been produced. Presented here is an example and supporting work to illustrate entanglement of logical qubits and how to interpret results of a simple quantum algorithm using them.

CHAPTER 2

BACKGROUND

2.1 HISTORICAL CONTEXT

Since the times of ancient philosophers, existential questions have fallen into two broader categories. Namely, there are questions regarding the meaning or purpose of reality, and there are questions regarding the nature of reality. As people began to unravel nature's rules, classical physics dominated the way the world was perceived. By applying classical physics and logistics principles the theory of computation was born. Classical computation traces its roots to almost one hundred years ago to Alonzo Church and Alan Turing. In 1936 and 1937, Church and Turing independently arrived at what would now become known as the Church-Turing thesis [2, 3]. This thesis and its supporting concepts have formed the foundation of most aspects of classical computing.

Around the same time the theory of computation was forming, great advancements in quantum physics were being made. In the 1920's and 1930's, famous scientists such as Werner Heisenberg and Paul Dirac were building upon foundational research made in the late 1800's by Michael Farday, Henrich Hertz, Max Planck, and others. Radical ideas such as wave-partical duality were being analyzed, calling into question many long-standing beliefs.

Physical computers were developing in leaps and bounds in the mid 1900's, and overcoming various engineering challenges presented by the technologies of the time to try and make a robust Turing-type machine. In an attempt to meet demands for large-scale computing without any errors, scientists began to analyze error correcting techniques. In 1950, Richard Hamming published a seminal work on error correcting codes that expanded on and and mathematically categorized different methods of ensuring data correctness [4]. Hamming's work on defining systematic codes is still in use today, and provides a starting point for topics

presented in following sections.

The computing and quantum physics disciplines continued in parallel for a time, before physicist Richard Feynman changed everything with a 1982 paper asking if quantum physics can truly be simulated via classical computation [5]. By breaking down quantum mechanics in terms of quantum states, Feynman argued that natural physics could never be truly simulated with a traditional "universal" Turing Machine! The very nature of atomic particles are continuous and analog, and can never be fully captured in a discrete set of tape symbols without losing some level of precision. Feynman concluded a classical universal device simply cannot represent the results of quantum mechanics accurately [5]. This reasoning seems to be a natural evolution of Turing's stipulation that his computing machine be operated against "finite" numbers or functions (naturally a bijection of sorts) [3]. Figure 2.1 below shows an illustration of a rudimentary understanding of this concept by showing discrete samplings of a continuous function. Note that in-between each discrete sampling there is potentially information lost.

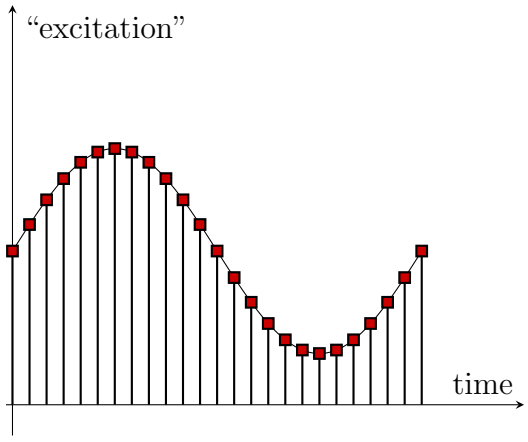

Figure 2.1: Discrete Sampling of a Continuous Function over Time

2.2 QUANTUM MECHANICS

There is one distinctive difference emphasized between classical and quantum mechanics. While studying classical mechanics, one is primarily concerned with everyday objects and the properties thereof such as kinematics, etc. In observing objects in this frame of reference, everything exists at a specific place at a particular time. I argue that every individual experiences reality phenomenologically differently. This might at first seem an extraneous note beyond the scope of this writeup, but its implications overlap into the quantum world quickly. When dealing with quantum mechanics, one is concerned of how physics behaves at the sub-molecular level with things such as atoms, electrons, photons, etc. With this frame of reference, objects exist in probabilistic superposition of multiple places at the same time [6, 1]! At this very small level of detail, classical mechanics does not apply as one's intuition may expect and interesting behaviors begin to arise. The famous thought experiment that begins to illustrate this concept is Schrödinger's Cat, where it was argued that under the experiment's conditions the cat was in a superposition of both alive and dead simultaneously.

Quantum mechanics at its core is the representation of particles using wave function mathematics [7]. Normal one dimensional mechanical wave functions can usually be written as the following:

$$y = (x, t) \tag{2.1}$$

Equation 2.1 has y representing the current "position" (or displacement) from equilibrium at time t, at a parallel line position x from the origin. Waves are always sinusoidal in nature, or a series of sinusoidal smaller waves [7]. Since particles exist in three dimensional space, we usually take x, y, and z coordinates into consideration. Common notation for a quantum wave function for a free particle (no resistance) uses the capital Greek letter psi, Ψ.

$$\Psi = (x, y, z, t) \tag{2.2}$$

Since this follows the general notation in the next sections, we will maintain this Ψ

standard of a quantum wave function before diverging to Hebrew letters in Section 4.3. Without diving deep into quantum physics, the Schrödinger Equation is presented next for completeness. This equation was developed by Erwin Schrödinger in 1926 and can be thought of as a decompilation of sorts of the quantum wave function. Not included here are the transitional steps from the traditional wave equation to the Schrödinger Equation, nor is the overall energy of the system explicitly defined.

$$\Psi_1 = i\hbar\frac{\partial\Psi_0(x,y,z,t)}{\partial t} \tag{2.3}$$

For our purposes, the most important element to note with regards to Equation 2.3 is the inclusion of the imaginary unit, $i = \sqrt{-1}$. The reduced Planck's constant, $\hbar = h/2\pi$, is also noteworthy. Planck's constant can be thought of as a fundamental constant used as a building block for other constants in physics. Since we are dealing with sub-molecular particles this type of constant makes sense to be included. The rest of the equation generalizes what one expects it should, where the partial derivatives of the dimensional coordinates with respect to time can be understood as self-evident. Therefore, ψ_1 is the application of this function at one additional "time click" against ψ_0. This generalization becomes important later, but do not get lost in this formula. For now, just understand that a quantum wave function has an imaginary component, thus making it complex in nature. This concept lends itself to a better understanding of the next section. More information on the Schrödinger Equation can be found in almost any modern university physics textbook.

Why do we write wave functions in this way? To begin to understand how computer scientists have devised ways to leverage these quantum physics principles for meaningful computations, we can further generalize our current understanding of a wave function down to a series of quantum states. Consider then a quantum wave function Ψ. At time t_0, Ψ's quantum state is Ψ_0, and at time t_1 its quantum state is in ψ_1. However, we will see that just knowing x, y, z, and ψ_0 is not enough information to classically compute ψ_1.

2.3 THE QUBIT

Making the jump from quantum mechanics to quantum computing is not all that daunting when considering quantum states in terms of Ψ. Computer scientists are interested in leveraging the complex nature of these quantum states and exploiting them for algorithmic speedups. Similar to classical computers, quantum computers consist of manipulations of a state machine. But only quantum computers can manipulate quantum states as compared to solely classical states. Quantum computers do not use regular classical bits that can either take a value of zero or one (usually implemented in reality with transistors). Instead, quantum computers use *qubits* which can be in a superposition of both zero and one simultaneously [8, 9, 1]. There are a variety of different types of physical qubit implementations including cold trapped ions [10], nuclear magnetic resonance (NMR) [11], and the more common superconducting methods [12, 13].

An excellent way to illustrate the nature of a universal qubit (regardless of physical implementation method) is that of a Bloch Sphere [1]. Figure 2.1 below shows how these qubits can have a type of "spin" indicating a direction in which they may be considered "pointing."

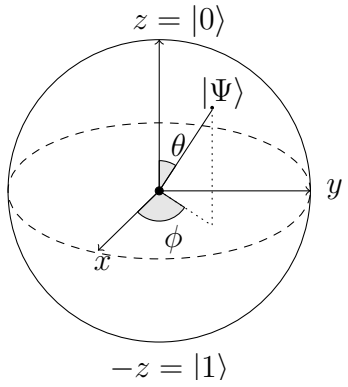

Figure 2.2: Bloch sphere representation of a qubit, adapted from page 161 of [1].

What can be inferred from the Bloch Sphere image above is how a qubit can have a value

of zero, one, or something in-between. The classical zero value correlates to the $|0\rangle$ state and the classical one value correlates to the $|1\rangle$ state. An in-between state is inferred as $|\Psi\rangle$, harkening back to our wave function idea. The $|\rangle$ symbols refer to *Braket* notation, of which will be detailed momentarily. Additionally, values in the positive or negative x direction are considered having a positive or negative *phase*. But let us not concern ourselves with the phase measurement for now. To begin, $|0\rangle$ and $|1\rangle$ are just vectors expanded on as shown.

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2.4}$$

This let's us combine all of the information presented thus far together to form the below generalizations of a qubit's state.

$$|\Psi\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \alpha\,|0\rangle + \beta\,|1\rangle = \alpha \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \beta \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2.5}$$

Similar to the Schrödinger Cat thought experiment, these qubits exhibit this type of superposition behavior until they are measured. Upon measurement, they collapse to either a $|0\rangle$ or $|1\rangle$ state! They can only be in this superposition state as long as they are unobserved. Due to the need to square the imaginary component to result in real values, the probability the qubit will collapse to $|0\rangle$ is $|\alpha|^2$ and $|1\rangle$ is $|\beta|^2$. Therefore, α and β can be thought of as probability amplitudes following $|\alpha|^2 + |\beta|^2 = 1$. What follows is that if a qubit has an α value of $\left(1/\sqrt{2}\right)$ and a β value of $\left(1/\sqrt{2}\right)$ then the qubit will collpase into state $|0\rangle$ or $|1\rangle$ with equal probability (a 50/50 coin flip)!

Note also the angles θ and ϕ may help define the $|\Psi\rangle$ state in a potentially easier to conceptualize format than vector notation. Things begin to get interesting when the spin value is neither directly up or directly down, but in some superposition state somewhere between these two values. This intermediate spin can be characterized by two trigonometric

angles, θ and ϕ. Through this method, it can be deduced that $0 \leq \phi < 2\pi$ and $0 \leq \theta < \pi$. For reasons beyond the scope of this introduction, only the $|1\rangle$ element is unfluenced by the imaginary component, due to a "global phase shift" [1]. Note also angle ϕ is entwined only with the imaginary component as well, as this can be thought of as our "third dimension" along the Bloch sphere. Figure 2.7 is included here for completeness, but will not be relied on for much of the analysis.

$$|\Psi\rangle = |q(\theta, \phi)\rangle = cos\frac{\theta}{2}|0\rangle + e^{i\phi}sin\frac{\theta}{2}|1\rangle \tag{2.6}$$

These generalizations and formulae hold true for qubits. Not covered here are qutrits or qunits, where the Bloch sphere is no longer divided into two extreme values, but three or n extreme values respectively. Qunit machines are not widely implemented in practice, although they are theoretically useful in reducing the number of fundamental particles needed to manipulate the desired quantum state for specific use cases [14]. This document will always assume the particles we are dealing with are represented as qubits, thereby only having the two extreme values.

2.4 MULTIPLE QUBITS

Representing a single qubit state with this type of Braket vector notation in and of itself is relatively uninteresting. One of the most prevalent means of quantum speedup involves the use of multiple qubits. Of critical importance here is that one can have multiple qubits in use in a quantum computing system, but they may or may not be necessarily *entangled*. Before looking into how this math works out, review the linear algebra concept of the tensor product as illustrated in Formula 2.7. Here there are two separate $|0\rangle$ valued qubits smashed together, but they are not *entangled*. Section 3.3 deals more with entanglement, but for now we will look at what happens with multiple qubits in the same system. Equations 2.7 and 2.8 utilize the *tensor* product, denoted $\otimes$.

$$|00\rangle = |0\rangle \otimes |0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ \\ 0 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.7}$$

Similarly, it follows that the two qubit value $|01\rangle$ can be represented as the below.

$$|01\rangle = |0\rangle \otimes |1\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ \\ 0 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \tag{2.8}$$

With two qubits, the total possible states (the resulting state vector) is 2^2. With three qubits, the output vector covers 2^3 possible states. Expanding on this, we understand that n qubits yield a potential state-space covering 2^n possible outcomes. This here gives the first whisper of how quantum computers can exponentially increase the potential result space, known as the *Hilbert Space*, with a fewer amount of computational steps when compared to a classical exponential algorithm. For n generalized qubits, we can illustrate the output space as in Equation 2.9.

$$|x_0 x_1 \ldots x_{n-1}\rangle = |x_0\rangle \otimes |x_1\rangle \otimes \ldots \otimes |x_{n-1}\rangle$$

$$= \begin{bmatrix} \alpha_{x_0} \\ \beta_{x_0} \end{bmatrix} \otimes \begin{bmatrix} \alpha_{x_1} \\ \beta_{x_1} \end{bmatrix} \otimes \ldots \otimes \begin{bmatrix} \alpha_{x_{n-1}} \\ \beta_{x_{n-1}} \end{bmatrix} = \begin{bmatrix} (\alpha_{x_0}\alpha_{x_1} \ldots \alpha_{x_{n-1}}) \\ (\alpha_{x_0}\alpha_{x_1} \ldots \beta_{x_{n-1}}) \\ \vdots \\ (\beta_{x_0}\beta_{x_1} \ldots \alpha_{x_{n-1}}) \\ (\beta_{x_0}\beta_{x_1} \ldots \beta_{x_{n-1}}) \end{bmatrix} \tag{2.9}$$

Equation 2.9 results in a single column, n-dimensional matrix or vector. Following the tensor product order of operations shows how an n number of two-state qubits can result in an 2^n-dimensional output state.

A word of caution is prudent to mention here regarding multiple qubits and how they operate with gates and logic as presented in Section 3.2. When dealing with multiple qubits and logic gates, it is often prudent to leverage what is known as a Kronecker product to appropriately perform the computations [15]. The Kronecker product is a specialization of the tensor product. This is not elaborated on here, but the reader is encouraged to research this topic for a more thorough understanding.

CHAPTER 3

THEORY OF QUANTUM COMPUTATION

Even with this understanding of quantum computing mathematics, the scope of problems able to be solved has not changed. Certain types of combinatorial problems can benefit from quantum speedup, but this not alter their purview at all [8]. More simply put, quantum computers are not magic and do not solve fundamental thought experiments such as the halting problem. When a qubit is measured, its state collapses and the quantum system effectively halts. When is the right time to measure and halt the operation? There are no definitive answers to this questions, but estimations and probabilities are deduced and formulated [16].

3.1 QUANTUM TURING MACHINES

Before describing the nature of a Quantum Turing Machine (QTM), it is critical to understand the *Unitary* concept and operations against the quantum state. Unitary operations are reversible and are key components to portraying and manipulating the unobservable quantum state(s) [1]. Unitary matrices take the form

$$UU^{\dagger} = U^{\dagger}U = I \tag{3.1}$$

where I is the identity matrix, and the † symbol is "dagger" notation for the conjugate transpose (also sometimes called a Hermitian transpose) of a matrix. The state of the quantum system evolves via unitary operations. Thinking back to our brief introduction to Schrödinger's equation, we can think of $|\psi_0\rangle$ as a quantum system at time t_0, and $|\psi_1\rangle$ as the state of the same system at time t_1 after the application of a unitary operation. This yields an understanding of $|\psi_1\rangle = \mathrm{U}\ |\psi_0\rangle$. The state of a quantum system is $|\Psi\rangle$, regardless of timestamp, and belongs to a "Hilbert Space" which is a complex inner product space

usually denoted $\mathcal{H}$. To further illustrate this point, suppose you have a QTM Q, consisting of n qubits which then form a state $|\Psi\rangle$ within $\mathcal{H}$, and having 2^n components of its state vector similar to Equation 2.9. Each element in that vector is itself a state space, which can take on different values depending on which Unitary operations have operated on the system at any point in a quantum algorithm.

This whole idea may seem like a trivial discussion point, but it is reiterated consistently in the literature as to its criticality [6, 5, 9]. Unitary operations by definition must be reversible. This means that any evolutional step of the quantum system must be able to go back to its previous state by a similar yet opposite de-evolutional step. The next section will discuss in detail different types of Unitary operations. Keep in mind that these operations can be reversed by applying the appropriate $U^\dagger$ operation.

QTM's follow a similar definition of a classical Turing Machine, with caveats in the definitions of its parameters [17]. The below definition has been pieced together to form a better cohesion with the other material in this document.

$$QTM = (\mathcal{H}, \Sigma, \Gamma, U, \mathcal{H}_0, \square, F) \tag{3.2}$$

Equation 3.2 gives a general definition of a QTM where $\mathcal{H}$ is a set of all possible states for the system as mentioned above. Σ is an input alphabet, such as only allowing $|0\rangle$ or $|1\rangle$ initialized qubits to begin with (therefore this can be classically visualized). Γ is the tape alphabet (a subset of $\mathcal{H}$). U can be written as $(\mathcal{H} - F) * U$ which can be thought of as any appropriate Unitary transitions within $\mathcal{H}$. $\mathcal{H}_0$ is the start state in $\mathcal{H}$ and F is the set of final possible states in $\mathcal{H}$. The $\square$ symbol is thought of as an "empty" tape symbol and is usually implied due to being confined within $\mathcal{H}$, but can also be understood as as an Identity operation (generally no effect).

This definition of a QTM does not offer much more than a comparison against classical Turing machines. The more common method of representing a specific QTM is by a Quantum

Circuit Diagram (QCD) [18] as is used in all subsequent sections of this document. These diagrams can help illustrate the flow of the overall state of the system by representing the interactions between qubits as well as imposed quantum gates. For a more detailed and formal definition of a QTM see Bernstein's and Vazirani's 1997 work [15].

3.2 GATES AND LOGIC

Similar to classical bits, qubits can be operated against using quantum gates. Gates can be chained together in various combinations to form logical operations to try and coerce a desired output state. Quantum gates can all be thought of as Unitary operations. Thinking back to our linear algebra primer, and how qubits can be represented as vectors or matrices, we begin to see how some of these operations unfold.

For example, take qubit q_0 as having a spin value of $|1\rangle$. This means that its matrix representation must be $\binom{0}{1}$. How would one go about applying a negation of q_0 to result in $q_0' = |0\rangle$? Similar to a classical "NOT" gate, there exists a Unitary operation commonly referred to as a Pauli-X gate [1]. This can be represented mathematically as a 2x2 matrix as illustrated below.

$$Xq_0 = X \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{3.3}$$

Feel free to convince yourself by working through the linear algebra. Note that these are standard matrix multiplications and not tensor products as discussed before. Also of note is that the X operation is representative of having its matrix "before" the target state if reading from left to right. It can be thought of as a type of function, taking the input state as an argument. If the α and β values of a qubit are not zero's or one's, then α and β are simply flipped (the new α becomes the old β value, and vice versa). The Pauli-X operation

is a member of the standard Pauli matrices, all of which manipulate a qubit's spin value as one might expect across the X, Y, and Z dimensions of the Bloch Sphere.

The second type of main introductory gate operation is the *Hadamard* gate. This gate takes any qubit initialized into state $|0\rangle$ or $|1\rangle$ and places it into a 50/50 superposition of both of these states. Recall from section 2.3 that a qubit with an α value of $\left(1/\sqrt{2}\right)$ and a β value of $\left(1/\sqrt{2}\right)$ will collapse into a $|0\rangle$ or $|1\rangle$ state with equal probability. Let us say that we are given two separate qubits, q_0 and q_1 with $|0\rangle$ and $|1\rangle$ states respectively as in Equations 3.4 and 3.5. Applying a Hadamard operation as follows yields our 50/50 coin flip state. This state becomes very important for various quantum computations.

$$Hq_0 = H \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} \tag{3.4}$$

Interestingly, if we use a different example qubit, q_1, having a value of $|1\rangle$ instead, we still result in a coin flip, but sign information is preserved!

$$Hq_1 = H \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix} \tag{3.5}$$

Compare equations 3.4 and 3.5. Note that each of their resulting α and β values still maintain the property of $|\alpha|^2 + |\beta|^2 = 1$ even though q_1's β value has a negative sign. Herein lies another whisper of quantum complexity.

There are other quantum gates covered in various materials. Summarized below are the Unitary matrices we have covered or alluded to thus far, and are sufficient for understanding subsequent sections for now.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{3.6}$$

For more meaningful computations, certain gates operating against one qubit may only be triggered on input from another qubit. The most straightforward example of this is the Controlled NOT (CNOT) gate. The CNOT will apply an X gate to a target qubit if and only if a secondary "source" qubit has a value of $|1\rangle$. This becomes interesting in that one cannot measure the the source qubit without collapsing the system. The best way to visualize this concept is through the use of a Quantum Circuit Diagram (QCD) showing the CNOT operation.

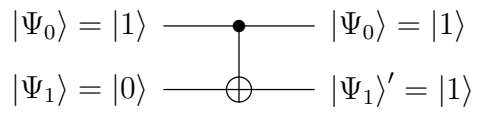

Figure 3.1: QCD of CNOT. Logic flows from left to right.

In Figure 3.1 we have two qubits starting with states $|\Psi_0\rangle$ and $|\Psi_1\rangle$[1]. The qubit with state $|\Psi_1\rangle$ has a Pauli-X gate operated against it if the qubit with state $|\Psi_0\rangle$ has a value of $|1\rangle$. For example, assume $|\Psi_0\rangle$ does indeed have a value of $|1\rangle$ and $|\Psi_1\rangle$ starts with a value of $|0\rangle$ as shown. It then follows that the CNOT operation would flip $|\Psi_1\rangle$ to its opposite value of $|1\rangle$ captured in $|\Psi_1\rangle'$ as seen in Figure 3.2.

This CNOT operation can also be represented via matrix multiplication. Combining examples as outlined in Figures 3.1 and Equation 3.3 with their surrounding paragraphs help lend an understanding to this concept. The CNOT gate is represented below in Equation 3.7 and shows a source qubit with value $|1\rangle$ flipping a target qubit from a $|0\rangle$ to a $|1\rangle$.

[1]When illustrating QCD's it is generally acceptable to use qubit names and their state values interchangeably. For example, assume qubit q_0 is the qubit with state $|\Psi_0\rangle$ in Figure 3.1, it is acceptable that the identifier "q_0" is not explicitly listed in the diagram. Similarly, it is generally correct to write the assignment of a state to an identifier i.e. $q_0 = |\Psi_0\rangle$.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = |11\rangle \tag{3.7}$$

The final quantum logic gate prudent to mention here is the *Toffoli* gate. This gate is simply a CNOT gate with multiple inputs or "source/control" qubits. The target qubit (indicated by the $\oplus$) is only flipped if both source qubits (indicated by the $\bullet$) collapse to a $|1\rangle$. This operation is utilized in further sections for more complex algorithms.

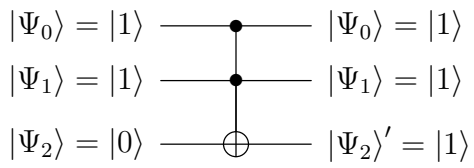

Figure 3.2: QCD of a Toffoli gate.

The Toffoli matrix representation follows similarly and is shown in Equation 3.8. The left-most matrix is representing the operation against the multiplicand is the state of our *three* qubits. Note that the Identity matrix is seeded into a large portion here, and the CNOT matrix we covered earlier is only acting upon the portion of $\mathcal{H}$ that is appropriate (the portion of the space influenced by the qubit beginning with a state value of $|\Psi_2\rangle$).

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = |111\rangle \tag{3.8}$$

3.3 ENTANGLEMENT

In 1964, a paper was published by John Bell which built upon previous works of Albert Einstein and others that filled in gaps in quantum mechanics regarding what was previously known as the "hidden variable" [19]. This work became a building block of our modern understanding of entanglement and *non-locality*, which is the understanding that if two or more quantum particles are properly entangled, then a measurement of one influences the measurement of the other(s) regardless of space covered between them. In a letter to Max Born in 1947, Einstein famously described this phenomena as "spooky action at a distance."

With two or more qubits, this possibility of entanglement arises. Quantum entanglement occurs when two particles interact with each other in such a way that each particle influences the others' "spin angle" and they can no longer be described independently [9, 1]. A common phrase one might use that could help illustrate this idea is "the whole is more than the sum of its parts." An example is if the particles that are used for the qubit implementation have an electromagnetic field, it is not much of a jump to accept that they can influence each other electromagnetically. For entangled qubits, the measurement of one particle not only collapses itself to an extreme value, but the other particle as well. This property is a cornerstone of

what makes quantum computers unique in their offerings for computational speedup through a type of inherent parallelism.

Let us begin with the most simple forms of quantum entanglement now known as the Bell State. This state can be derived from two qubits, a Hadamard gate, and a CNOT gate. Diving right into a visual representation, see Figure 3.2 for the creation of such a state.

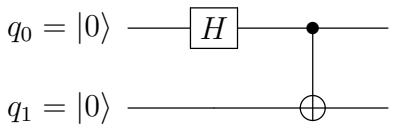

Figure 3.3: QCD of two physical qubits forming the Bell State.

The diagram in Figure 3.2 shows us that the first qubit, q_0 is first placed into a superposition state with the Hadamard operation. Following this, a CNOT gate is operated against q_1 as the target, with q_0 being the source. Recall that the CNOT operation only changes the output of the target qubit if the source/input qubit collapses to a value of $|1\rangle$. But what happens when the source qubit is in a superposition state of both $|0\rangle$ and $|1\rangle$? The target qubit *also* gets placed into a superposition state of $|0\rangle$ or $|1\rangle$ depending on which value the source is in! But the source's value cannot be known until it is measured, meaning that the target qubit also cannot be known until it is measured. Thus, both qubits are in their own superposition state but dependent on one another! For example, if at the end of the computation suppose that only q_1 is measured. Through our understanding, we can then infer and extrapolate that q_0 *must* be the same value as q_1 (and vice versa)! If q_0 measures to a $|0\rangle$ then the CNOT operation would not change the value of q_1, resulting in it maintaining its $|0\rangle$ value. If q_1 measures to a $|1\rangle$ then the CNOT operation flips the value of q_1, resulting in it flipping its value to a $|1\rangle$.

The Bell State naturally evolves to more qubits in a similar fashion, where one qubit is placed into superposition and subsequent qubits are "chained" together and toggle from a

$|0\rangle$ to a $|1\rangle$ state based on the previous' qubit's value. Figure 3.4 shows a QCD of such a state, and its output is ideally either $|0000\rangle$ or $|1111\rangle$ (with approximately equal probability). However, Chapter 4 discusses how this is not always the case in practice.

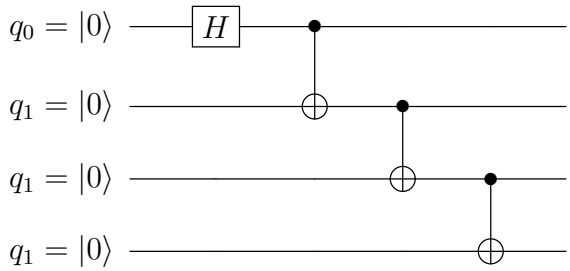

Figure 3.4: QCD of four physical qubits forming the Bell State.

CHAPTER 4

ERROR DETECTION AND CORRECTION

In the late 1980's and early 1990's, the stage had been set for the topic of Quantum Error Correction (QEC). Much of the foundational physics, mathematics, and engineering for executing quantum computations had been laid. Quantum systems were extremely fragile, and susceptible to different types of noise and decoherence. Scientists set out to devise ways to account for these widely apparent errors by not only detecting when they occurred, but to correct them as they happened. Known for some time had been the idea of the "no-cloning theorem" which stated that any quantum bit could not be observed or measured without collapsing it into its base or excited state - thereby losing its superposition [20]. It was being found that the quantum system was nearly impossible to completely isolate from its surrounding environment. Qubits were being "measured" in certain ways by the outside environment and were getting introduced to unnecessary collapsing errors.

There are three main types of errors that can be exhibited against a physical qubit, and they are the (1) bit flip, (2) sign flip, or (3) the bit and sign flip together [1]. The sign flip can also be known as a phase flip or shift [21]. Going back to Figure 2.2, the sign flip is thought of as flipping the direction along the x axis. We have already worked through bit flip examples by illustrating the Pauli-X gate. An example of this bit flip error occurring is if the system behaved as if an unintended X gate were applied to some qubit(s). The sign flip is harder to visualize due to its complex nature (recall the imaginary numbers used to represent these systems). A general example of a sign flip is for a $+|\Psi\rangle$ to flip to a $-|\Psi\rangle$. For now, it is sufficient to assume that a phase flip is equivalent to a Pauli-Z gate operating against a qubit.

4.1 HAMMING CODES

In 1950, Richard W. Hamming made significant contributions in codifying and demonstrating error correction schemes for classical computers [4]. This section describes some of these mathematics, and then works through an example error correction code. If you feel sufficiently comfortable with these topics, you may be better suited to skipping to the next section.

The general idea behind what are now known as Hamming Codes is to encode a given word (i.e. a string of classical bits) into a longer word of the same language in such a way that the added information can help detect and correct any errors to the original word should they appear during computation, etc. For example, consider the [7,4] Hamming Code [4]. The first parameter in the bracket is the length of the *encoded* word and the second parameter is the length of the *source* word (the word to be protected against errors). There are an additional three bits added known as *parity bits* that are calculated from the source word in a clever way as follows. Sometimes this code is written as [7,4,3] where the last parameter is known as the *distance*, k, where $k-1$ is the maximum amount of errors the code can tolerate before it is unable to properly detect and correct errors.

The [7,4] Hamming Code is created by taking an input binary string, $x_3x_2x_1x_0$, and combining it with parity bits $p_4p_2p_1$. A parity bit p_3 is not included in this notation, because this way lends itself better to thinking in terms of binary counting, and this scheme operates somewhat akin to a binary search. In this example, the words and parity bits are formed as shown in Equation 4.1. The XOR operation here is as one would expect a binary exclusive-or operation to occur.

$$\begin{gathered}\text{Source Word: } x_3x_2x_1x_0 \\ p_1 = x_3\ XOR\ x_2\ XOR\ x_0 \\ p_2 = x_3\ XOR\ x_1\ XOR\ x_0 \\ p_4 = x_2\ XOR\ x_1\ XOR\ x_0 \\ \text{Encoded Word: } p_1p_2x_3p_4x_2x_1x_0\end{gathered} \tag{4.1}$$

A common way to illustrate this concept is with a Venn diagram. This illustration was not included in Hamming's original paper, but can be found in almost many modern introduction to this topic. Presented below in Figure 4.1 is such a diagram for the [7,4] Hamming Code following the concept laid out in Equation 4.1.

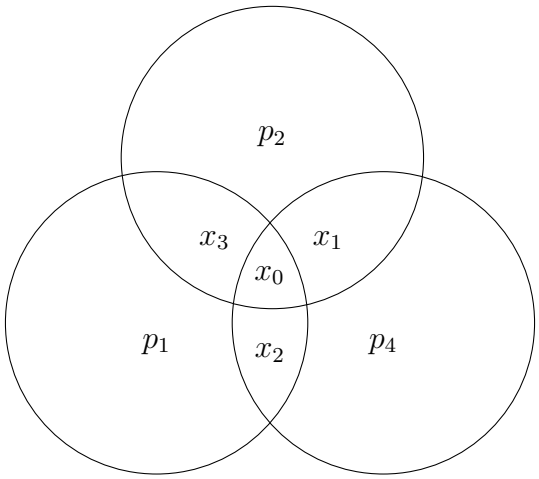

Figure 4.1: Venn diagram of the [7,4] Hamming Code.

Finally, with the added parity bits the source word can be checked for errors. Suppose an error occurred on bit x_1 and its value flipped from a 0 to a 1 or a 1 to a 0. This would cause the values of p_2 and p_4 to no longer align (be equal to) a re-computation of their values. Put another way, if the XOR operations were performed again, these two parity bits would no longer equal what they were originally set to. The algorithm would then detect that since both p_2 and p_4 both agree but do not equal what they were supposed to based on the source

word, it can accurately detect the input bit, x_1 as having flipped. An exercise left to the reader is to confirm the reverse is true as well, if a parity bit experienced an error it can be assured that it can be detected with accuracy based on a re-computation of values.

While not explored in extreme detail, this example is meant to prime the reader for the next section. The ideas of adding additional parity bits and forming encoding schemes to account for errors in clever ways is the foundation upon which qubit error detection is built. In fact, the Steane code explored at length in Appendix C is similar to the [7,4] Hamming Code as we will see.

4.2 SINGLE QUBIT ERRORS

Asher Peres is one of the first to publish a work regarding QEC [9]. This work produced an error correction scheme for single types of errors and its modernized version is summarized below. Similar to Hamming Codes, the idea is to spread a single qubit's state information across multiple qubits. If an error occurs on one of the qubits, a type of majority vote is used to determine what the value should actually be. This is similar to how disk striping and some RAID configurations are understood. Figure 4.2 below shows such an encoding and decoding of a single qubit quantum state, $|\Psi\rangle$ being distributed amongst three physical qubits, q_0, q_1, and q_2. Following this, we denote these three physical qubits make up one *logical* qubit.

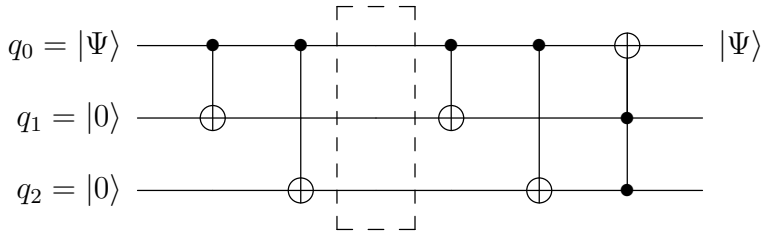

Figure 4.2: QCD of a bit flip error correction code.

The dotted lines in Figure 7.1 symbolize a potentially noisy or error-prone environment. This code can protect a single bit-flip error. We are assuming the initialization and operations

prior to the section captured in the dotted lines is error-free, even though it is demonstrated later that this is not always a safe assumption. For now, let us suppose a bit flip error happens on q_1 in this dotted line section. We can visualize this with an X gate. The idea here is that even with this single bit-flip error occurring, the encoded $|\Psi\rangle$ state is able to be safely decoded.

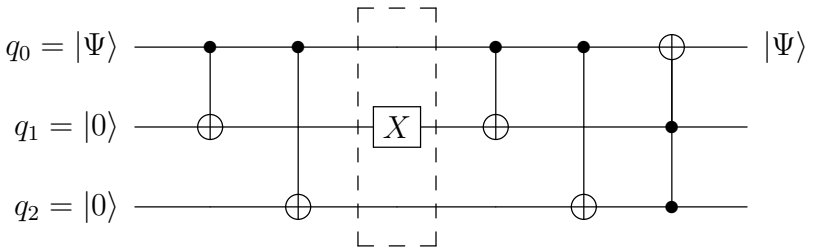

Figure 4.3: QCD of a bit flip error correction code with error present.

We can illustrate this via matrix multiplication. For simplicity, we will assume that $|\Psi\rangle$ has a value of $|1\rangle$, but note that any complex α and β states are preserved similarly. Equation 4.2 below shows the three qubits forming a 2^n state space, upon which the CNOT operations embedded into a Unitary matrix are executed. After the CNOT operations are performed an X error is introduced into the q_1 subset of the Hilbert Space (our $2^n x 2^n$ matrix). Then follows the decoding steps of two CNOT gates and a Toffoli gate operating back onto q_0. Note that q_0's output value is safely maintained to the appropriate $|1\rangle$ value even with this error introduced!

If $|\Psi\rangle = |1\rangle$ for this example then Equation 4.1 shows how adding two additional $|0\rangle$ initialized qubits into the representation would follow. A matrix transpose T operation is used here to conserve space.

$$|\Psi 00\rangle = |100\rangle = |10\rangle \otimes |0\rangle = |1\rangle \otimes |00\rangle = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}^T \tag{4.2}$$

Equation 4.1 is split up in this way to demonstrate how the tensor product can be split

up accordingly amongst these qubits. This lends itself to better understanding the CNOT operations. Applying the first CNOT operation yields the below linear algebra computation in Equation 4.2. Recall back to Section 3.2 and in particular Equation 3.3 that discusses how the order of operations for matrix multiplication gates flows intuitively "backwards" when compared to the flow of the logic expressed in the QCD.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = |11\rangle \tag{4.3}$$

The output in Equation 4.3 covers q_0 and q_1 values after the first CNOT gate illustrated in Figure 4.3. Following this logic, we use this output as input into the next CNOT operation, but we only care about the q_0 value because it is the source qubit in determining whether or not the target q_2's value will flip. This results in the same multiplication we just computed! Where the output of $q0$ CNOT q_2 is also $|11\rangle$. Therefore, combining all three qubit values *before* the noisy channel is $|111\rangle$.

Entering the noisy channel, our example now supposes that a bit flip error occurs on qubit q_1. Our precious quantum state has now incorrectly changed from $|111\rangle$ to $|101\rangle$ at this time. All hope is not lost though! Because we have encoded this information across three physical qubits we use the following gates in the QCD to perform a type of "majority vote" and decide once and for all that qubit q_0 is indeed maintaining its value of $|1\rangle$. An exercise left to the reader is to determine what would happen if the bit flip error occurred on q_0 itself giving it a value of $|0\rangle$ in the noisy channel. This scheme flips the value back to a $|1\rangle$ prior to reading the output at the end.

4.3 TRANSVERSAL GATES

In the previous section we discussed the bit flip error detection and correction code. This is the most simple of codes, and can only correct for one instance of one type of error amongst one logical qubit consisting of three physical qubits. This current section assumes some knowledge of the seven qubit Steane error correction code which will protect a logical qubit from a bit flip and/or a phase flip occurring in any of its seven physical qubits. More information on the Steane QEC scheme is found in Appendix C, and a novel method of using ancillae qubits is presented there. Once we have logical qubits from this method, the question then becomes what can we do with them?

The idea presented here is that of *transversal* gates, which can be thought of as "logical qubit gates" that act on logical qubits themselves. These transversal gates in actuality consist of a collection of regular quantum gates acting against the physical qubits making up the logical qubits but in an organized way such that quantum state information is not sacrificed or compromised. This concept and its limitations are explored in a 2009 paper by Bryan Eastin and Emanuel Knill's [22]. Caveats and notes regarding these limitations are noted after the following example.

Consider two logical qubits, **א** and **ב**. These logical qubits are being represented with the Hebrew letters aleph, **א**, and bet, **ב**, to avoid confusion with physical qubits. Suppose we want to perform a CNOT operation against these two logical qubits such that when logical qubit **א** measures a $|1\rangle$ state then the value of logical qubit **ב** is flipped as illustrated in the QCD in Figure 4.3.

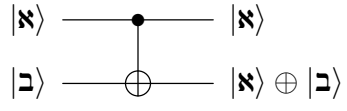

Figure 4.4: Logical qubit operation.

Logical qubit א is made up of seven physical qubits, q_0 to q_6. Logical qubit ב is made up of seven physical qubits, q_7 to q_{13}. The quantum state $|$א$\rangle$ has been "broken" into sub-states $|$א$_0\rangle$ to $|$א$_6\rangle$ with each sub-state applied to one of the physical qubits making up א (it may help to think of this as "projecting" the information into a higher dimension Hilbert Space). Similarly, the quantum state $|$ב$\rangle$ has been "broken" into sub-states $|$ב$_0\rangle$ to $|$ב$_6\rangle$ with each sub-state applied to one of the physical qubits making up ב.

To form the transversal CNOT gate, the physical qubit gates cascade in the manner as shown in Figure 4.5. Figures 4.5 and 4.5 are logically equivalent. This transversal gate is appropriate for the Steane encoding of logical qubits (see Appendix C for more information).

Not all gate operations can be applied in this way however. In fact, what is now known as the Eastin-Knill theorem states that there is no non-trivial encoding for logical qubits that are able to implement all quantum logic gates transversally [22]. For example, the Steane code cannot transversally implement the T gate (see Appendix A) [23]. Potential solutions around this might be to encode and decode with different QEC schemes periodically in the middle of a quantum algorithm, depending on which transversal operation you are needing at the time. This obviously instills a large amount of complexity and circuit depth overhead and comes with its own set of problems.

4.4 ENTANGLED LOGICAL QUBITS

This section intends to bring all of the information together especially Sections 3.3 and 4.3. Referring back to Section 3.3, let us revisit the concept of quantum entanglement. Recall that the definition of an entangled system is that it cannot be "separated" into its constituent parts [1]. What this means is that by measuring a portion of the entangled system you know with certainty information about the state of the rest of the entangled system without measuring it at all! For example, take two entangled physical qubits q_0 and q_1. If one were to measure q_0, then without measuring q_1 you would already know its value or spin angle (and

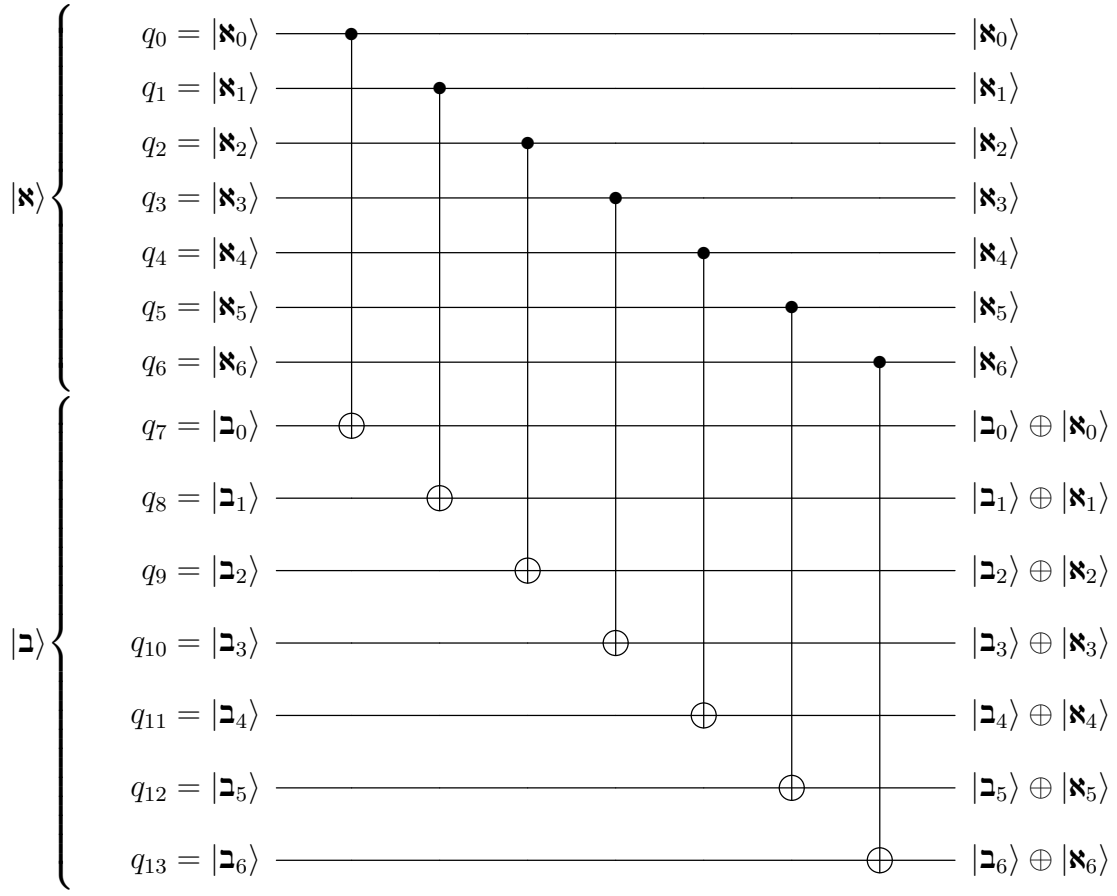

Figure 4.5: Logical qubit transversal CNOT operation.

vice versa)! What follows is the expansion of this concept onto logical qubits. We again show the Bell State in Figure 6.1, but this time with two logical qubits instead of two physical qubits.

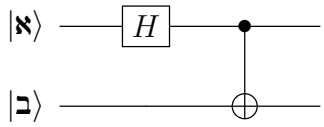

Figure 4.6: QCD of two logical qubits forming the Bell State.

Hebrew letters are used again for reinforcement that these logical qubits are different from physical qubits. To maintain consistency (and correctness), these logical qubits א and ב are formed using the Steane code. Logical qubits prefer to operate using transversal gates. In section 4.3 the transversal CNOT gate was illustrated. A logical Hadamard gate is similar, in that a Hadamard operation is applied against all of the physical qubits making up the logical qubit. Therefore, to illustrate a logical Bell State the expanded diagram would look like it does in Figure 6.2.

The experiments conducted primarily revolve around implementing this logical bell state, injecting errors, and watching the quantum program detect and correct them. Without any errors present, this Bell State of two logical qubits results in outputs of $|00\rangle$ and $|11\rangle$ of approximately 1:1 ratio. For example, if the program is ran 100 times, one would expect an output of $|00\rangle$ approximately 50 times, and $|11\rangle$ approximately 50 times. If any of the readouts resulted in either of the other states ($|01\rangle$ or $|10\rangle$) then we can be certain that some amount of errors occurred beyond that which this scheme was able to account for. Each logical qubit using this Steane QEC method is able to detect and correct a single bit flip and/or a single phase flip amongst any of its physical qubits. Similarly, the ancilla qubits needed for the Steane method must be operated against without error, which is not always guaranteed. See Appendix X for more information on the ancilla qubit information.

4.5 TOPOLOGICAL AND LATTICE CODES

This brief section is included here for completeness, but not expounded on in detail. A different QEC technique is that of encoding qubits into lattice-type structures. This alternative scheme operates on a different, yet adjacent, type of quantum computer than that which has been discussed thus far, and uses *topological* qubits defined using *anyons* rather than traditional qubits [24, 25]. These anyons are manipulated in such a way that they can "swap" places with each other over time, and depending on how the lattice is

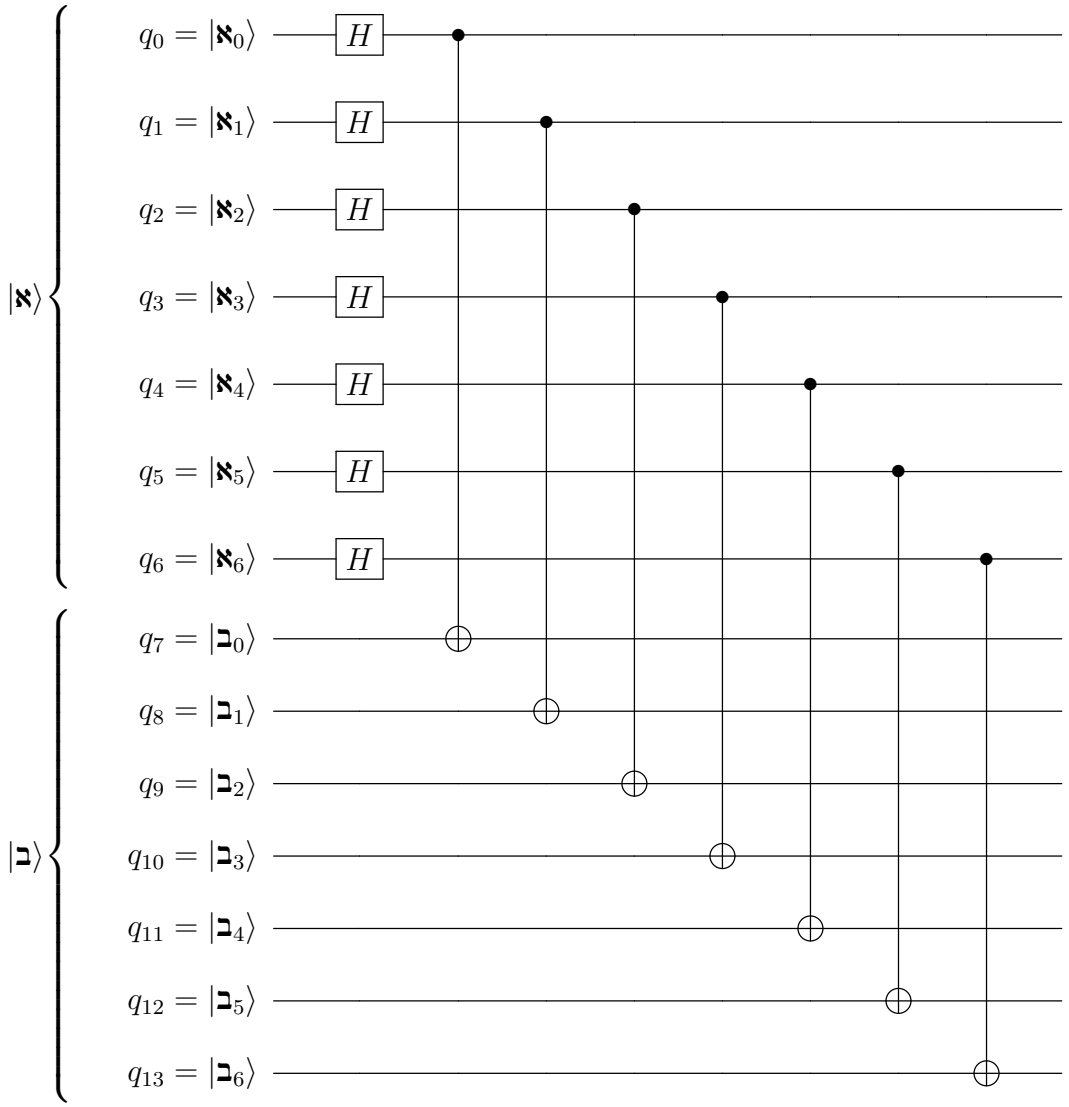

Figure 4.7: Logical qubit Bell State with Transversal Gates.

structured determines the paths in which they can swap [25]. There are tradeoffs associated with implementing quantum algorithms in this as opposed to the traditional "spin" methods discussed above, and they are not covered here. For more information on this topic and the alleged resilience to certain errors these types of systems offer, the reader is encouraged to read Wille Lahtinen's and Jiannis Pachos's 2017 publication [26]. Indeed, these systems may indeed be a valid next step to mitigating engineering challenges as presented in Section 6.3.

CHAPTER 5

THRESHOLDS AND COMPLEXITY

What makes a fault tolerant quantum computer? The ability of any quantum computer to compute quantum algorithms correctly without being negatively influenced by errors may indeed be an impracticality due to the inherent noisy nature of these systems. Does this mean that all hope is lost for any practical usage of these systems? Not quite. The usefulness of noisy quantum systems depends on what problem one is trying to solve or how one aims to measure success. For example, it has been argued that today's extant Noisy Intermediate-Scale Quantum (NISQ) technologies which consist of a few hundred noisy qubits can outperform classical computers for specific tasks [27]. This chapter begins by discussing how quantum computers' errors can be quantified with relation to different error correction techniques. After this is introduced, an analysis of time and space complexity of these systems and their offerings is given, providing a baseline upon which the next chapter will compare experimental results.

5.1 ERROR THRESHOLDS

Let us first devise some rudimentary *a priori* understandings of error thresholds in quantum systems. Suppose you are given a quantum computer Q of which you are wishing to analyze whether or not your particular quantum program will return results correctly. For simplicity, let us assume that Q is resilient to all errors except bit-flip errors, and that our definition of successful program execution means the readout of the system is not negatively influenced by any errors in any noisy channels. Q experiences errors on a single qubit with probability p per each computational time step (we will soon see why this is a broad generalization). Without using any QEC techniques or encoding physical qubits into logical qubits, assume that your quantum program requires n physical qubits, and t

computational time steps to execute. Yet an additional assumption here is that out of scope for this discussion is the negation of errors. For example, if Q experiences an error against qubit q_i at time t_j, and at a further time t_{j+x} this same qubit experiences an additional error that would theoretically "cancel out" the previous error that occurred at t_j, it is just assumed that q_i cannot be trusted at all, and contributes to a readout error. A general visualization of what such a QCD may look like is illustrated in Figure 5.1.

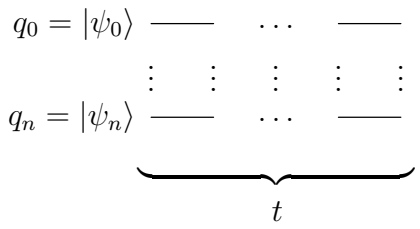

Figure 5.1: Theoretical Quantum Computer Q.

From this definition it follows that Q can perform approximately $\frac{1}{np}$ computations until experiencing an error. Therefore, our only hope of having a general confidence that Q runs our quantum program correctly is if $t < \frac{1}{np}$. For example, if our quantum program requires four qubits, and $p = 0.01$, then we could assume to experience some type of error in around twenty-five steps of computation.

Now assume you have a second quantum computer, Q′. Q′ only experiences bit-flip errors on a single physical qubit with probability of p per computational time step similar to Q, but Q′ performs all needed computations transversally using logical qubits. For general readability, assume the Steane error correction method is used to form logical qubits, each requiring seven physical qubits each. To run our quantum program requiring n qubits, Q′ requires these qubits to be logical qubits according this scheme. Therefore, the total number of physical qubits required to run the same quantum program on Q′ is $7n$ (the number of ancilla qubits needed is considered negligible for this example). This is certainly a significant

increase in terms of space by today's standards, as physical quantum computers today operate on the scale of a few hundred real qubits [27].

But now examine the probability of our program failing to execute correctly on Q′. The Steane QEC method enables the system to detect and correct any single physical error in a logical qubit at the expense of some constant number of computational steps. This constant overhead of computational steps to encode and decode a single logical qubit we will call C, and is considered error-resilient. Therefore, the overall overhead for running our program requiring n logical qubits is nC, which just results in another constant C. Because our program on Q′ now uses logical qubits that are protected against single bit-flip errors, what this means is that for the output of our program to read incorrectly we must experience more than one error in a single computational step in a single logical qubit.

Following these definitions, we can then assert that if such a QEC method was implemented with Q' then we can have a general confidence that Q' runs our quantum program correctly if $t < \frac{7}{np} + C$. Going back to the previous example of a quantum program requiring four qubits (logical or physical) and the probability of an error occurring at any given timestamp $p = 0.01$, we now have approximately 175 steps of computation we can reasonably assume to execute with any errors occurring able to be detected and corrected as part of an added cost in C. It is critical to note here that the constant overhead for the encoding and decoding is not always negligible in this way though.

5.2 COMPLEXITY AND BENCHMARKING

In 1997, Peter Shor published a paper that that caused quite a stir regarding polynomial-time quantum computers offering a solution to a prime factorization problem using quantum computers [28]. Forming semi-prime numbers from prime numbers is a key ingredient to certain encryption standards that bank on brute-forcing their secrets being so great in computational complexity being it is impractical with modern hardware. If quantum

computers could overcome challenges presented by their noisy nature, here it was argued that they would be able to efficiently provide a subroutine towards this problem. Some of these key points are worth mentioning here and have been alluded to throughout this document. What follows is an abstract attempt to provide the reader with a type of intuition into these ideas.

First and foremost, the idea that n qubits can form a computational space consisting of 2^n possible states is a key ingredient into reducing complexity. With this point alone we have an argument that there is potentially an exponential type of speedup waiting to be explored. Secondly, the idea of entanglement discussed at length in Section 3.3 lends itself to the idea of a single qubit "projecting" its information onto another qubit, thereby further increasing portions of the potential state space. Finally, using creative methods to eliminate portions of a quantum systems possible Hilbert Space as viable, one can form a type of logarithmic approach to a spatial subset. It is not critical for our introductory purposes to understand the fine details here, but to begin to be open to the idea of using these computers differently than traditional methods.

All the way back in Section 2.1 it was mentioned that quantum computations deal with a type of analog information. Then in Section 2.3 it was shown that the information quantum computers operate with are qubits. Chapter 3 demonstrated combining various qubits and performing logical operations. Following this progression, it is understood that even with "analog" quantum information, the *computations* themselves are not analog. QTM's should be seen as consisting of discrete operations, and as such are able to compared and contrasted to certain extents with classical computing methods [15]. This section further analyzes time and space complexity of operating with quantum computers by first comparing them against classical computers and then makes a case for solving certain problems more efficiently than classical computers.

Classical computers can measure their speed with floating-point operations per second

(FLOPS). These operations consist of addition, subtraction, multiplication, and division of very large or very small numbers [29]. The decimal point of these numbers "floats" (somewhat akin to scientific notation) in such a way that an attempt to optimize the memory and computational steps that are needed has been made at the hardware level. The details of these operations are not covered here, but suffice to say it should be self-evident that certain calculations are more expensive such as multiplication requiring more computational steps than addition. These differences in computations can reduce the clarity of this metric. Nevertheless, companies still publish this information for classical Central Processing Units (CPU's) and GPU's. For an example, one of Intel's more recent I7-12700T consumer-grade processors is is advertised to be capable of 268.8 Gigaflops (GFLOP) (one billion floating-point operations per second) [30]. This may feel like such a large amount of computational resources that newer technologies such as quantum computing may never catch up.

As compared to classical FLOPS/GFLOPS, Quantum computers generally have an advertised Circuit Layout Operations Per Second (CLOPS) metric performed by a Quantum Processing Unit (QPU). Thinking back to Section 3.2, we again analyze the idea of quantum gates in a quantum circuit. The number of qubits and the number of connections between them are obviously strong spatial indicators, and these make up the proposed benchmark known as *quantum volume* (QV) [31]. CLOPS is a measurement of manipulating QV and includes the speed in which classical computers are able to interface with a quantum computer, as without this type of interaction the system is meaningless. Interestingly, CLOPS is not generally improved by enhancements to the classical computing architecture, but rather the "bus" between the two systems [31, 32]. As such, the CLOPS metric pairs nicely as the temporal element to benchmarking these systems.

CHAPTER 6

EXPERIMENT DETAILS

In the late 2000's and and 2010's the quantum computing field went through an explosive level of growth, but there was a general uncertainty over what the technology was capable of. Companies like IBM and Google were building larger and larger physical quantum computers, and companies like DWave were selling consumer-grade systems to industry. In hindsight, it seems many claims as to the near-term goals of this technology around this timeframe may have been over-aggressive. Great strides in the mathematics have been made in the past couple of decades, but the engineering of the physical systems remains largely inadequate for groundbreaking operations. Challenges related to physical quantum computers include their small number of available qubits, the architecture in which their qubits are able to be entangled, coherence times, susceptibility to noise, and general availability. In an attempt to to better understand the science and theory behind these machines, simulating their properties using Graphics Processing Units (GPU's) was proven useful. After showing successful simulations, this chapter discusses results from a real quantum computer from IBM located in Brisbane, Australia. The following sections detail some of these efforts, the results obtained, and an analysis on what some of the next steps could be.

6.1 GRAPHICS PROCESSING UNITS

The experimentation began using GPU's and Nvidia's Compute Unified Device Architecture (CUDA). The reasoning for starting in this way was to better understand the material before using physical quantum computers. Additionally, the intention is to make this work approachable and easily accessible to a wider audience, and GPU's are more readily available than physical quantum computers. In hindsight, this may have been a mistake in starting with this implementation for reasons covered in Section 6.3. But this hindsight was only

gained after extensive research and building upon the foundation that the GPU systems enabled.

GPU's are not quantum computers. As such, to perform any type of quantum algorithm on them they must be configured to run a *simulation* of a quantum environment. We have seen in previous sections the extensive matrix multiplications and manipulations that can be performed with just a handful of qubits. It does not seem like much of a leap to implement these types of algorithms with GPU's as they are well-suited to these types of tasks. But a critical component that GPU's are missing is the susceptibility to errors that appear with physical quantum systems. Implementing QEC techniques with GPU simulations is essentially wasted computations and resources except for research purposes. Injecting different errors into the environment manually with differing probabilities yielded different results, of which will be discussed momentarily.

How would one begin to store a quantum state in GPU memory? Let us go back all the way to Section 2.2 and 2.3 and revisit the idea of representing a qubit in terms of a Bloch Sphere. We eventually arrived at Equation 2.6 showing an imaginary component to a qubit. Therefore, we confidently state here that qubits are represented via complex numbers which usually take the form $a + bi$ where a is a real component, and b is a constant multiplied by the imaginary unit i. Reducing Equation 2.6 to this form allows us to store a single qubit's information in terms of a and b, where i is implied. Following this logic, all that is left is determining what level of precision is desired for a and b. This granularity and memory space is adjustable based on different CUDA parameters.

Starting with Nvidia's cuQuantum library, these experiments began quickly with a C++ styled implementation that gave a much more low-level access to what the GPU was computing and gave better insight into the Hilbert space and Unitary operations [33]. This in combination with the related software Nvidia's Aer simulator provided the quick tools to drive low level results quickly. Shown below is a code snippet of a basic initialization of the

quantum state and the Hadamard matrix. This is provided to provide a sort of insight into how these types of data objects would look in memory.

```
cuDoubleComplex h_sv[] = {{0.5773502, 0.0}, {0.8164965, 0.0}};
cuDoubleComplex h_sv_result[] = {{0.986, 0.0}, {-0.169, 0.0}};

// Hadamard approximated
cuDoubleComplex matrix[] =
{{0.7071067, 0.0}, { 0.7071067, 0.0},
 {0.7071067, 0.0}, {-0.7071067, 0.0}};
```

This code initializes one qubit's worth of information into object *h_sv* where the *_sv* nomenclature is understood to indicate a *state vector* implementation. This document and all testing does not cover the more complex *tensor network* implementations. For more information on this code and a link to the GitHub repository, please see Appendix F. This object contains an array with two elements, the qubit's α and β values, where each of these elements is a set of its own two components. The first component of each of the α and β elements is the real component, and the second relates to the imaginary portion, thus forming the complex numbers. The second elements in the inner arrays are set to zero values in this example, indicating there are no imaginary components. This is done for easier understanding, but these can be large floating point numbers as well.

Note the granularity of the Hadamard approximation. When this matrix is applied against the qubit state info in the *h_sv* object then the result is supposed to approximately equal that of *h_sv_result*. Let us verify this quickly in Equation 6.1. The imaginary components are left in to help illustrate these are complex numbers being stored in memory.

$$\begin{bmatrix} 0.7071067 + 0i & 0.7071067 + 0i \\ 0.7071067 + 0i & -0.7071067 + 0i \end{bmatrix} \begin{bmatrix} 0.5773502 + 0i \\ 0.8164965 + 0i \end{bmatrix} \approx \begin{bmatrix} 0.986 + 0i \\ -0.169 + 0i \end{bmatrix} \tag{6.1}$$

After initializing these values, they are offloaded onto the GPU where instructions are sent to manipulate the Hilbert Space accordingly by applying various Unitary operations. Once the computation is complete, the result is sent back across the bus to the main computer's memory and accessed by the CPU for the output to be displayed. This model follows similarly for next topic, Qiskit, but much of the lower level operations are masked and inaccessible to the user.

A big takeaway here is the representation of the complex numbers, and how these numbers grow exponentially. Here is discussed the level of precision and how this chews up memory. Tradeoffs are associated with loss of precision, but can be mitigated with more cycles (i.e. hyperparameters).

Shortly after this cuQuantum proof of concept, a GPU-enabled version of IBM's Python library, Qiskit, was used for the remainder of the experimentation. This Qiskit library was compiled manually using cuQuantum flags so that the GPU would still be enabled similarly. This library was good for two main reasons. Firstly, it was much faster to code quantum circuits using this method as the syntax was greatly simplified. Secondly, this library can be used for both simulations via classical computers, as well as passing instructions off to physical quantum computers from IBM's cloud API's.

Below is an example of our favorite Bell State implemented in Qiskit. In the code modules that were produced as part of this work, this function is called from a driver if the "simple" physical bell state is desired (no logical qubits or operations). Other modules were built for the various other QCD's including the ones for QEC and logical qubits. Nevertheless, this sample is provided here to illustrate the syntax and to allow the reader to begin to visualize how the more complex circuits might be built in code.

```
from qiskit import *

def new_simple_bell_circuit():
        q = QuantumRegister(2,'q')
        c = ClassicalRegister(2,'c')

        circuit = QuantumCircuit(q,c)

        circuit.h(q[0]) # initialize superposition
        circuit.cx(q[0],q[1]) # entanglement - bell state

        circuit.measure(q[0],c[0])
        circuit.measure(q[1],c[1])

        return circuit
```

More information on the code used for this effort can be found at the GitHub repository referenced at Appendix F.

6.2 IBM QUANTUM COMPUTER

There is no such thing as a human-operable "pure" quantum computer. All quantum computers have some element of classical control mechanisms intertwined into their architecture. Normally, classical computers input their desired data into the quantum system, coerce the quantum system to execute the desired computations, and then read out the result at the appropriate time [5]. For some instances this may be done in a looping mechanism where consistent inputs or tweaking is necessary. This section discusses at length running

our Bell State on a real quantum computer provided by IBM using its Eagle architecture. More information on this offering is found through IBM's website [34].

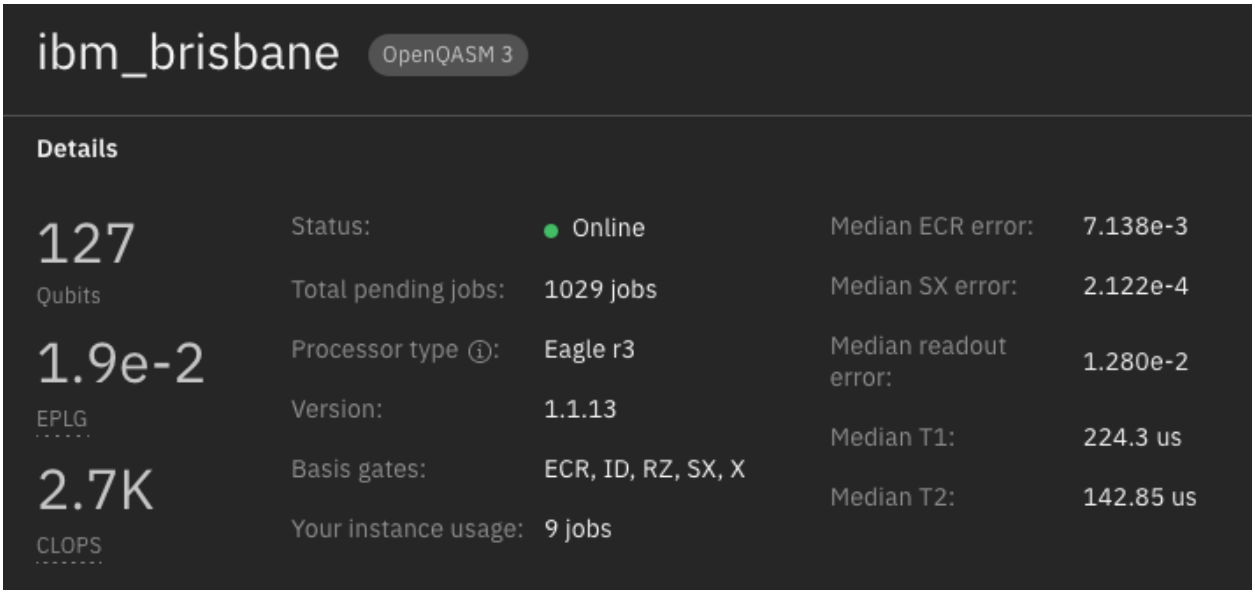

Figure 6.1: Information on IBM's quantum computer in Brisbane, Australia.

When a quantum program is built in a high-level language like Qiskit, it undergoes a process known as *transpilation* before it is able to execute on a physical quantum computer. The term "transpile" is a combination of "compile" and "translate" in that the high-level quantum circuit is translated into different yet logically equivalent gates that the coomputer is able to execute. This step is very similar to a traditional understanding of compilation where a high-level programming language is transformed into machine code, but in this case the input and output language is the same. But, the gates are very different in terms of operations and circuit depth as shown in the upcoming example and in the next section. Herein lies the crux of an issue where many of these QEC schemes begin to stop offering physical benefits outside of research in that physical quantum systems may or may not be currently engineered to optimize these operations.

Let us begin with our familiar Bell State implemented with two physical qubits. Hearken

all the way back to Figure 3.3 and take note of how many gate operations it takes to form this state with an "ideal" quantum computer. There is one Hadamard gate, and one CNOT gate. For quantum computers that have the potential to experience an error during a gate operation, it is desirable to keep circuit depth low. Unfortunately, many physical quantum computers do not have the ability to implement certain gates in actuality. For example, the IBM Brisbane model does not have the ability to execute "pure" Hadamard and CNOT gates to form this Bell State. Instead, the transpiler knows that to create the logical equivalence of this state it needs to work with the gate operations it has access to. As such, the transpiled QCD from Figure 3.3 to what is able to run on the IBM Brisbane computer is as follows.

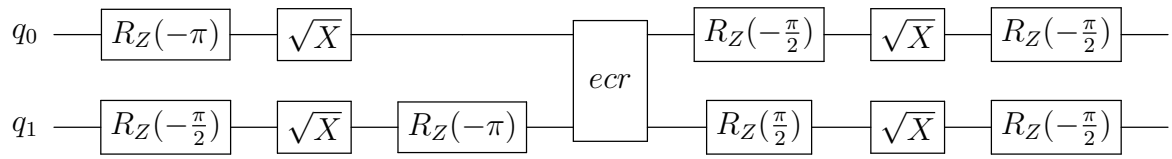

Figure 6.2: Transpiled QCD of two physical qubits forming the Bell State.

To demonstrate the correctness of this transpiled circuit, it is a good idea to work through at least a portion of the appropriate matrix multiplications these gates represent. After arriving at an output state, the result can then compared to the "ideal" circuit's output state. We begin by introducing this concept, but leave the majority of the computations up to the reader if they so wish to compute them.

Referring back to Figure 6.4 above, suppose both q_0 and q_1 are initialized with values $|0\rangle$ each, therefore the first operation to perform against each qubit is $R_z(-\pi)$ and $R_z(-\frac{pi}{2})$ respectively.

$$q_0 = R_z(-\pi)\,|0\rangle = \begin{bmatrix} e^{-i\frac{-\pi}{2}} & 0 \\ 0 & e^{i\frac{-\pi}{2}} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} e^{i\frac{\pi}{2}} \\ 0 \end{bmatrix}$$
$$q_1 = R_z(-\tfrac{\pi}{2})\,|0\rangle = \begin{bmatrix} e^{-i\frac{-\frac{\pi}{2}}{2}} & 0 \\ 0 & e^{i\frac{-\frac{\pi}{2}}{2}} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} e^{i\frac{\pi}{4}} \\ 0 \end{bmatrix} \tag{6.2}$$

After this first set of gates, now both q_0 and q_1 each undergo a $\sqrt{X}$ gate (sometimes denoted SX). Following Appendix A, and remembering to place this gate as the left-most operand, each qubit's state now becomes

$$q_0 = \tfrac{1}{2} \begin{bmatrix} 1+i & 1-i \\ 1-i & 1+i \end{bmatrix} \begin{bmatrix} e^{i\frac{\pi}{2}} \\ 0 \end{bmatrix} = \tfrac{1}{2} \begin{bmatrix} (1+i)e^{i\frac{\pi}{2}} \\ (1-i)e^{i\frac{\pi}{2}} \end{bmatrix}$$
$$q_1 = \tfrac{1}{2} \begin{bmatrix} 1+i & 1-i \\ 1-i & 1+i \end{bmatrix} \begin{bmatrix} e^{i\frac{\pi}{4}} \\ 0 \end{bmatrix} = \tfrac{1}{2} \begin{bmatrix} (1+i)e^{i\frac{\pi}{4}} \\ (1-i)e^{i\frac{\pi}{4}} \end{bmatrix} \tag{6.3}$$

From here, only q_1 has another $R_z(-\pi)$ gate operated on it. Giving it this value before the *ecr* gate.

$$q_1 = \begin{bmatrix} e^{-i\frac{-\pi}{2}} & 0 \\ 0 & e^{i\frac{-\pi}{2}} \end{bmatrix} \begin{bmatrix} \frac{1}{2}(1+i)e^{i\frac{\pi}{4}} \\ \frac{1}{2}(1-i)e^{i\frac{\pi}{4}} \end{bmatrix} = \tfrac{1}{2} \begin{bmatrix} e^{-i\frac{-\pi}{2}}(1+i)e^{i\frac{\pi}{4}} \\ e^{i\frac{-\pi}{2}}(1-i)e^{i\frac{\pi}{4}} \end{bmatrix}$$
$$= \tfrac{1}{2} \begin{bmatrix} ie^{i\frac{\pi}{4}}(1+i) \\ -ie^{i\frac{\pi}{4}}(1-i) \end{bmatrix} = \tfrac{1}{2} \begin{bmatrix} (-1+i)e^{i\frac{\pi}{4}} \\ (-1-i)e^{i\frac{\pi}{4}} \end{bmatrix} \tag{6.4}$$

Now both qubits q_0 and q_1 enter into an echoed cross-resonance (ecr) gate. This gate can be thought of as a fancy CNOT type of operation, but only operating on appropriately

transpiled qubits. Equation A.6 shows this gate by itself and how it relates to the Identity, X, and Y gates. Because this is a two qubit gate, the matrix representations of q_0 and q_1 are first tensored together similar to Equation 2.7 and then multiplied by the ecr matrix.

$$q_0q_1 = \frac{1}{\sqrt{2}}\begin{bmatrix} 0 & 1 & 0 & i \\ 1 & 0 & -i & 0 \\ 0 & i & 0 & 1 \\ -i & 0 & 1 & 0 \end{bmatrix} (\frac{1}{2})(\frac{1}{2}) \begin{bmatrix} ((1+i)e^{i\frac{\pi}{2}}) * ((-1+i)e^{i\frac{\pi}{4}}) \\ ((1+i)e^{i\frac{\pi}{2}}) * ((-1-i)e^{i\frac{\pi}{4}}) \\ ((1-i)e^{i\frac{\pi}{4}}) * ((-1+i)e^{i\frac{\pi}{4}}) \\ ((1-i)e^{i\frac{\pi}{4}}) * ((-1-i)e^{i\frac{\pi}{4}}) \end{bmatrix} \tag{6.5}$$

We discontinue the matrix computations here in the interest of conciseness having established the general cadence in which these multiplications are occuring. The state eventually evolves to what one expects, with various imaginary values cancelling each other out and assuming the desired superposition state.

$$q_0q_1 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{bmatrix} \tag{6.6}$$

Figure 6.3 gives basic information about this quantum computer, and also provides information on the advertised error parameters. Starting with the top left, we see that this computer offers 127 qubits, but these are not necessarily "general purpose" qubits. The EPLG value stands for the error per layered gate. The ECR value stands for "Echoed Cross-Resonance" gates - which are similar to CNOT gates as we have seen. SX is the squared root of X. T1 and T2 are together called "decoherence times." T1 is "energy relaxation" - in which a qubit randomly decays from a $|1\rangle$ to a $|0\rangle$ [31]. T2 errors are "dephasing" errors. For more information, the reader is encouraged to look into the IBM Quantum Platform [35].

CHAPTER 7

DATA AND RESULTS

What follows are a series of tables showing different error correction schemes being fed manual errors in a "noisy" channel similar to Figure 4.3. The Bell State is configured and then appropriate errors are generated with increasing probability to see how the schemes perform accordingly. Conclusions are drawn in the following chapter. After the section on manual errors has been presented, then follows results using an IBM noise model from an Eagle quantum processor. This latter data is generated from both GPU simulations and a physical quantum computer located in Brisbane, Australia.

7.1 MANUAL ERRORS

The following data tables are results obtained using the Python Qiskit library leveraging local GPU computations. The two physical or logical qubits were placed into a Bell State, and then their outputs were sampled to see if they correctly displayed the $|00\rangle$ or $|11\rangle$ states at approximately a 1:1 ratio. The experiment was ran one hundred times each for each tuple [1]. Significant overhead was needed for the manual error generation as compared to the next section due to the nature of the software library used. To manually generate errors with some probability, the entire QCD had to be destroyed and reconstructed on each iteration of a loop. During the QCD generation, the appropriate Pauli-X or Pauli-Z gates were placed into the noisy channel with their respective probabilities to simulate errors. Once there, the algorithm was allowed to proceed as normal to see how its output would measure.

[1] Qiskit often refers to the number of loops an experiment takes as the number of *shots* that were performed.

Manual Error's Against Two Logical Qubits with Bit Flip Encoding						
Bit Flip Probability	Errors	$\|00\rangle$	$\|01\rangle$	$\|10\rangle$	$\|11\rangle$	Sucess
0.0	0	58	0	0	42	100.0%
0.05	37	40	0	2	58	98.0%
0.1	55	42	1	2	55	97.0%
0.15	84	47	5	9	39	86.0%
0.2	126	42	11	8	39	81.0%
0.25	157	38	9	19	34	72.0%
0.3	190	32	22	19	27	59.0%
0.35	203	32	19	18	31	63.0%
0.4	249	16	31	22	31	47.0%
0.45	274	34	19	24	23	57.0%
0.5	297	27	29	16	28	55.0%
0.55	345	22	18	27	33	55.0%
0.6	330	22	30	15	33	55.0%
0.65	389	24	25	25	26	50.0%
0.7	412	45	15	11	29	74.0%
0.75	445	44	20	9	27	71.0%
0.8	494	43	7	7	43	86.0%
0.85	513	44	7	3	46	90.0%
0.9	540	43	5	3	49	92.0%
0.95	567	52	0	1	47	99.0%
1.0	600	54	0	0	46	100.0%

Table 7.1: Two Bit-Flip Encoded Logical Qubits on the A1289 Platform

Manual Errors Against Two Logical Qubits with Phase Flip Encoding						
Phase Flip Probability	Errors	$\|00\rangle$	$\|01\rangle$	$\|10\rangle$	$\|11\rangle$	Sucess
0.0	0	53	0	0	47	100.0%
0.05	39	51	2	1	46	97.0%
0.1	56	45	5	0	50	95.0%
0.15	86	52	2	6	40	92.0%
0.2	135	40	11	11	38	78.0%
0.25	158	35	17	11	37	72.0%
0.3	201	28	23	22	27	55.0%
0.35	208	32	18	19	31	63.0%
0.4	226	29	28	25	18	47.0%
0.45	276	30	25	16	29	59.0%
0.5	305	28	26	20	26	54.0%
0.55	330	23	30	25	22	45.0%
0.6	363	25	14	29	32	57.0%
0.65	360	27	29	21	23	50.0%
0.7	420	30	19	20	31	61.0%
0.75	459	46	13	11	30	76.0%
0.8	477	29	9	11	51	80.0%
0.85	516	49	6	7	38	87.0%
0.9	545	48	3	1	48	96.0%
0.95	575	45	2	0	53	98.0%
1.0	600	52	0	0	48	100.0%

Table 7.2: Two Phase-Flip Encoded Logical Qubits on the A1289 Platform

Manual Errors Against Two Logical Qubits with Shor Encoding						
Phase Flip Probability	Errors	$\|00\rangle$	$\|01\rangle$	$\|10\rangle$	$\|11\rangle$	Sucess
0.0	0	52	0	0	48	100.0%
0.05	85	54	5	4	37	91.0%
0.1	145	38	11	10	41	79.0%
0.15	258	27	15	21	37	64.0%
0.2	352	29	24	18	29	58.0%
0.25	457	38	17	27	18	56.0%
0.3	574	26	31	23	20	46.0%
0.35	678	23	21	35	21	44.0%
0.4	713	24	23	27	26	50.0%
0.45	812	25	24	30	21	46.0%
0.5	908	26	23	18	33	59.0%
0.55	975	28	22	27	23	51.0%
0.6	1094	24	17	31	28	52.0%
0.65	1180	23	29	30	18	41.0%
0.7	1266	32	20	32	16	48.0%
0.75	1356	26	22	26	26	52.0%
0.8	1438	27	22	24	27	54.0%
0.85	1528	30	22	16	32	62.0%
0.9	1649	39	6	10	45	84.0%
0.95	1714	46	5	6	43	89.0%
1.0	1800	43	0	0	57	100.0%

Table 7.3: Two Shor Encoded Logical Qubits with Phase Flip (Z) Errors Only

Manual Errors Against Two Logical Qubits with Steane Encoding						
Phase Flip Probability	Errors	$\|00\rangle$	$\|01\rangle$	$\|10\rangle$	$\|11\rangle$	Sucess
0.0	0	48	0	0	52	100.0%
0.01	23	51	3	0	46	97.0%
0.02	40	55	2	4	39	94.0%
0.03	59	46	2	3	49	95.0%
0.04	61	47	2	4	47	94.0%
0.05	82	49	4	9	38	87.0%
0.06	102	35	9	3	53	88.0%
0.07	125	50	10	7	33	83.0%
0.08	134	43	9	14	34	77.0%
0.09	170	31	14	16	39	70.0%
0.1	161	39	9	12	40	79.0%
0.11	188	34	15	13	38	72.0%
0.12	189	32	14	16	38	70.0%
0.13	233	27	17	19	37	64.0%
0.14	223	34	14	23	29	63.0%
0.15	257	33	22	19	26	59.0%
0.16	261	25	31	15	29	54.0%
0.17	278	21	20	28	31	52.0%
0.18	310	24	27	21	28	52.0%
0.19	309	19	31	14	36	55.0%
0.2	355	25	24	23	28	53.0%

Table 7.4: Two Steane Encoded Logical Qubits with Phase Flip (Z) Errors Only

Manual Errors Against Two Logical Qubits with Steane Encoding						
Phase Flip Probability	Errors	$\|00\rangle$	$\|01\rangle$	$\|10\rangle$	$\|11\rangle$	Sucess
0.0	0	48	0	0	52	100.0%
0.01	15	44	2	0	54	98.0%
0.02	31	52	1	2	45	97.0%
0.03	62	47	2	1	50	97.0%
0.04	75	48	1	4	47	95.0%
0.05	88	41	4	11	44	85.0%
0.06	79	37	3	9	51	88.0%
0.07	111	36	11	14	39	75.0%
0.08	138	33	10	19	38	71.0%
0.09	142	34	9	15	42	76.0%
0.1	162	27	20	13	40	67.0%
0.11	200	32	16	17	35	67.0%
0.12	192	36	16	13	35	71.0%
0.13	242	29	21	22	28	57.0%
0.14	240	26	20	16	38	64.0%
0.15	244	35	19	11	35	70.0%
0.16	268	30	22	18	30	60.0%
0.17	279	37	14	24	25	62.0%
0.18	300	26	23	24	27	53.0%
0.19	334	28	24	13	35	63.0%
0.2	355	34	22	17	27	61.0%

Table 7.5: Bell State with Steane Transversal Operations

Summarized from the tables above are the following graphs. These demonstrate that when accuracy is increased (i.e. the probability to generate errors in the noisy channel is decreased) then the success of the experiment samples increased (i.e. whether the qubit state correctly outputs a $|00\rangle$ or $|11\rangle$ state. It can be found interesting as the probability of errors approach one, the entire algorithm is essentially reversed, where the values that would have normally output $|00\rangle$ are now outputting $|11\rangle$ and vice versa. This essentially masks the issue, and a "sweet spot" exists between probability of 0.5 and probability of 1.0 for useful information. Note however that the Steane encoding does not display this full probability spread for brevity, and provides a deeper granularity into increasing success as accuracy increases.

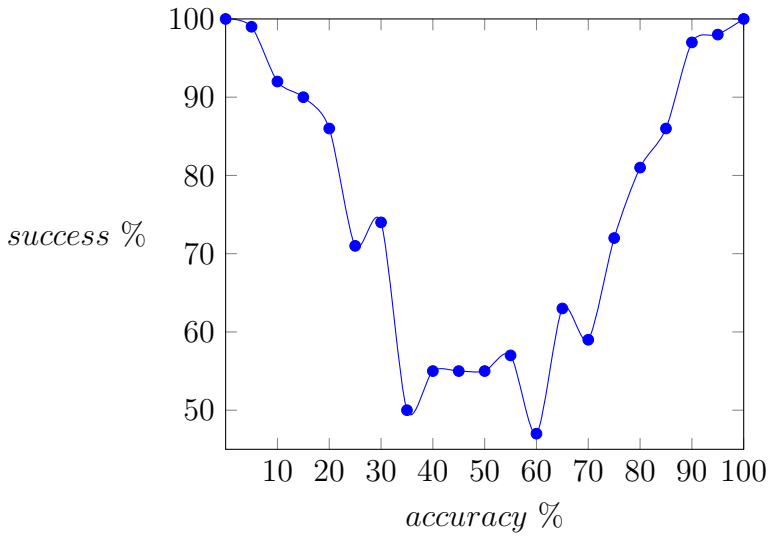

Figure 7.1: Bell State with Logical Qubits using Bit-Flip Encoding

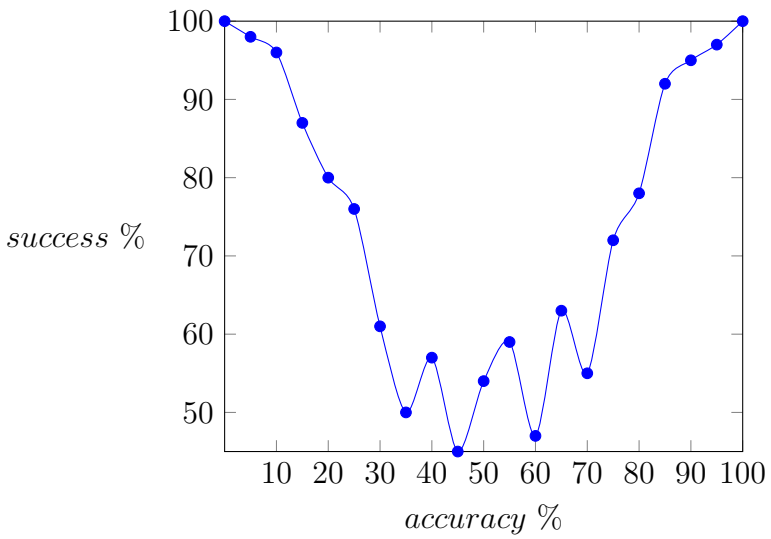

Figure 7.2: Bell State with Logical Qubits using Phase-Flip Encoding

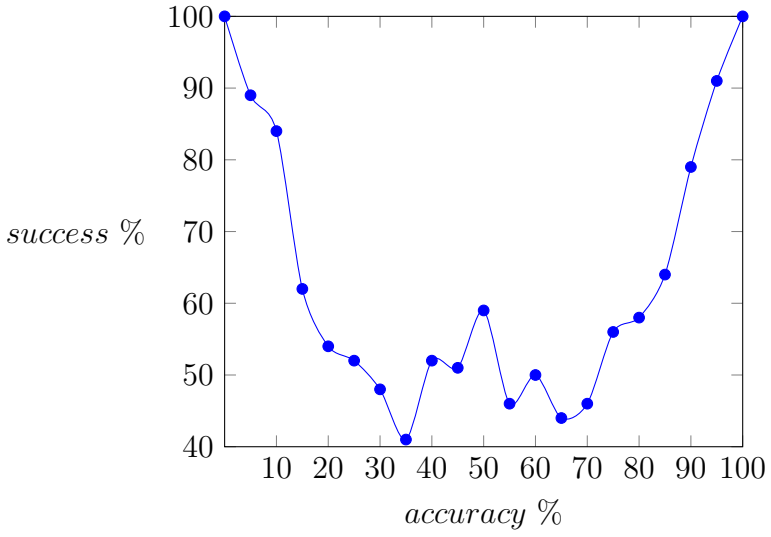

Figure 7.3: Bell State with Logical Qubits using Shor Encoding

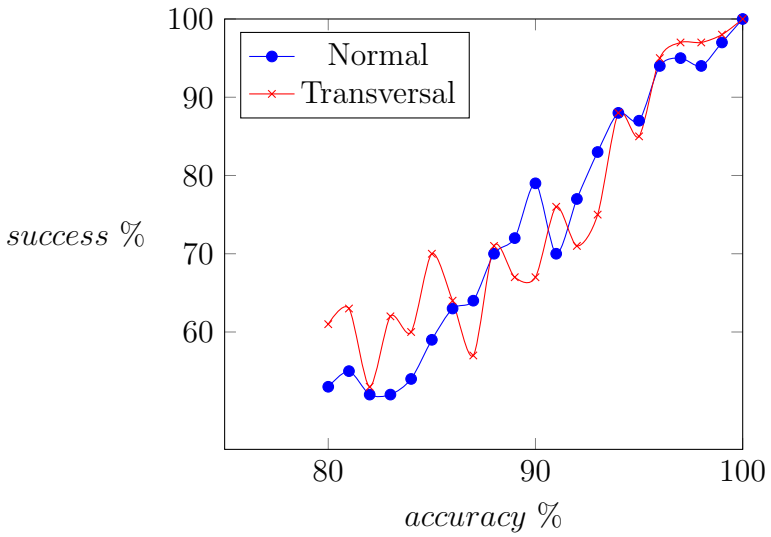

Figure 7.4: Bell State of Logical Qubits with Steane Encoding.

Figure 7.4 above can be considered a culmination of this research, demonstrating two entangled logical qubits operating roughly equivalently with transversal operations compared to "normal" operations. This idea lends credence to building deeper and more complex algorithms with QEC methods.

7.2 EAGLE PROCESSOR ERRORS

Output data from experimenting with IBM cloud services is presented below. Shown first is a baseline experiment using a noise model obtained from IBM mimicking their advertised values for their Eagle processor. Included in this matrix are results from the physical quantum computer located in Brisbane, Australia as well as GPU simulations.

Two Qubit Bell State with the IBM Brisbane Noise Model			
Environment	Qubit Configuration	Success %	Number of Gates
A1289	Two Physical	95.2%	4
A1289	Bit-Flip Encoding	92.3%	14
A1289	Phase-Flip Encoding	94.1%	26
A1289	Steane Encoding	87.3%	169
Campus Cluster	Two Physical	94.7%	4
Campus Cluster	Bit-Flip Encoding	92.7%	14
Campus Cluster	Phase-Flip Encoding	92.5%	26
Campus Cluster	Steane Encoding	89.9%	169
IBM Brisbane	Two Physical	93.2%	14
IBM Brisbane	Bit-Flip Encoding	79.5%	194
IBM Brisbane	Phase-Flip Encoding	87.7%	199
IBM Brisbane	Steane Encoding	N/A	N/A

Table 7.6: IBM Brisbane Noise Model Results

Drilling down into the first tuple from table 7.6 would yield something like represented in Table 7.7.

Details on Two Physical Qubits with A1289 and Brisbane Noise Model					
$\|00\rangle$	$\|01\rangle$	$\|10\rangle$	$\|11\rangle$	Total	Successful
498	23	25	454	1000	952

Table 7.7: Two Physical Qubits on the A1289 Platform

These results correspond strongly with the transpilation challenges mentioned in Section 6.2. Even with error correction algorithms running with no manual errors seeded, the noise of the physical system still generates less than ideal metrics. Of note here is that the "noisy channel" mentioned above is different in the physical machine, as each error is at the gate level. Certain gate operations are treated as "ideal" and error-resistant in the simulations, but in actuality, this is not the case with this real quantum computer.

CHAPTER 8

CONCLUSION

We have demonstrated here simulating entangled logical qubits using GPU's. More than this, the Bell State used to illustrate our entanglement was also ran against a physical quantum computer offered by IBM. Extensive background and introductory materials have been offered, and an argument for using these systems has been given.

In putting together this research something that became immediately apparent is the decline of accuracy in implementing logical qubits with the default configurations of IBM's physical quantum system. The transpilation process was an unforeseen roadblock that increased the amount of quantum logic gates needed to perform a logically equivalent quantum program by around fifty times. This in turn caused the probability of errors occurring to increase significantly and caused for a general decoherence of the system. Combined with an average queue time of roughly two hours to run a single program here with a free student license made debugging this challenging.

Besides this finding, simulating these systems without error or with a low noise model has shown promise in that circuit depth may theoretically be able to be increased for more complex algorithms beyond that of just initializing a Bell State. Furthermore, to strive towards this goal it is important to engage in architectural conversations in the future in how physical quantum systems are built. Engineering-focused implementations of logical qubits may help reduce general noise in the system if they are designed with this at the beginning. Noisy systems may be somewhat acceptable in the near term, but for any strong advancements to be made in this field requiring little to no error a general redesign may be called into question. Reducing transpilation overhead by thinking of "error-correction first" hardware design choices may offer support for error-sensitive use cases in the future.

REFERENCES

[1] N. S. Yanofsky and M. A. Mannucci, *Quantum computing for computer scientists*. New York : Cambridge University Press, OCLC: ocn212859032.

[2] A. Church, "A set of postulates for the foundation of logic," vol. 33, no. 2, p. 346, 204 citations (Crossref) [2023-03-14]. [Online]. Available: https://www.jstor.org/stable/1968337?origin=crossref

[3] A. M. Turing, "On computable numbers, with an application to the entscheidungsproblem," vol. s2-42, no. 1, pp. 230–265, 1877 citations (Crossref) [2023-03-14]. [Online]. Available: http://doi.wiley.com/10.1112/plms/s2-42.1.230

[4] R. W. Hamming, "Error detecting and error correcting codes," vol. 29, no. 2, pp. 147–160. [Online]. Available: https://ieeexplore.ieee.org/document/6772729

[5] R. P. Feynman, "Simulating physics with computers," vol. 21, no. 6, pp. 467–488, 4037 citations (Crossref) [2023-03-14]. [Online]. Available: http://link.springer.com/10.1007/BF02650179

[6] C. H. Bennett, "Logical reversibility of computation," vol. 17, no. 6, pp. 525–532, 2158 citations (Crossref) [2023-03-15]. [Online]. Available: http://ieeexplore.ieee.org/document/5391327/

[7] H. D. Young, R. A. Freedman, and A. L. Ford, *University Physics with Modern Physics: Sears and Zemansky's*, 13th ed. Pearson education, OCLC: 832678193.

[8] D. Deutsch, "Quantum theory, the church–turing principle and the universal quantum computer," vol. 400, no. 1818, pp. 97–117, 2225 citations (Crossref) [2023-03-14]. [Online]. Available: https://royalsocietypublishing.org/doi/10.1098/rspa.1985.0070

[9] A. Peres, "Reversible logic and quantum computers," vol. 32, no. 6, pp. 3266–3276, 547 citations (Crossref) [2023-03-14]. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.32.3266

[10] J. I. Cirac and P. Zoller, "Quantum computations with cold trapped ions," vol. 74, no. 20, pp. 4091–4094. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.74.4091

[11] D. G. Cory, M. D. Price, W. Maas, E. Knill, R. Laflamme, W. H. Zurek, T. F. Havel, and S. S. Somaroo, "Experimental quantum error correction," vol. 81, no. 10, pp. 2152–2155, 396 citations (Crossref) [2023-03-14]. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.81.2152

[12] W. D. Oliver and P. B. Welander, "Materials in superconducting quantum bits," vol. 38, no. 10, pp. 816–825, 145 citations (Crossref) [2023-03-14]. [Online]. Available: http://link.springer.com/10.1557/mrs.2013.229

[13] A. P. M. Place, L. V. H. Rodgers, P. Mundada, B. M. Smitham, M. Fitzpatrick, Z. Leng, A. Premkumar, J. Bryon, A. Vrajitoarea, S. Sussman, G. Cheng, T. Madhavan, H. K. Babla, X. H. Le, Y. Gang, B. Jäck, A. Gyenis, N. Yao, R. J. Cava, N. P. de Leon, and A. A. Houck, "New material platform for superconducting transmon qubits with coherence times exceeding 0.3 milliseconds," vol. 12, no. 1, p. 1779, 119 citations (Crossref) [2023-03-14]. [Online]. Available: https://www.nature.com/articles/s41467-021-22030-5

[14] N. Ba An, "Teleportation of two-quNit entanglement: Exploiting local resources," vol. 341, no. 1, pp. 9–14. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0375960105006134

[15] E. Bernstein and U. Vazirani, "Quantum complexity theory," vol. 26, no. 5, pp. 1411–1473, 781 citations (Crossref) [2023-03-14]. [Online]. Available: http://epubs.siam.org/doi/10.1137/S0097539796300921

[16] A. Hagar, *The Curse of the Open System.* Springer International Publishing, pp. 5–21. [Online]. Available: https://link.springer.com/10.1007/978-3-031-02514-3_2

[17] C. Moore and J. P. Crutchfield, "Quantum automata and quantum grammars," vol. 237, no. 1, pp. 275–306, 221 citations (Crossref) [2023-03-14]. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0304397598001911

[18] A. Chi-Chih Yao, "Quantum circuit complexity," in *Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science.* IEEE, pp. 352–361. [Online]. Available: http://ieeexplore.ieee.org/document/366852/

[19] J. S. Bell, "On the einstein podolsky rosen paradox," vol. 1, no. 3, pp. 195–200. [Online]. Available: https://link.aps.org/doi/10.1103/PhysicsPhysiqueFizika.1.195

[20] W. K. Wootters and W. H. Zurek, "A single quantum cannot be cloned," vol. 299, no. 5886, pp. 802–803. [Online]. Available: http://www.nature.com/articles/299802a0

[21] P. Benioff, "Quantum mechanical models of turing machines that dissipate no energy," vol. 48, no. 23, pp. 1581–1585. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.48.1581

[22] B. Eastin and E. Knill, "Restrictions on transversal encoded quantum gate sets," vol. 102, no. 11, p. 110502. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.102.110502

[23] E. T. Campbell, B. M. Terhal, and C. Vuillot, "Roads towards fault-tolerant universal quantum computation," vol. 549, no. 7671, pp. 172–179. [Online]. Available: https://www.nature.com/articles/nature23460

[24] S. B. Bravyi and A. Y. Kitaev, "Quantum codes on a lattice with boundary," publisher: arXiv Version Number: 1. [Online]. Available: https://arxiv.org/abs/quant-ph/9811052

[25] S. H. Simon, N. E. Bonesteel, M. H. Freedman, N. Petrovic, and L. Hormozi, "Topological quantum computing with only one mobile quasiparticle," vol. 96, no. 7, p. 070503. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.96.070503

[26] V. Lahtinen and J. Pachos, "A short introduction to topological quantum computation," vol. 3, no. 3, p. 021. [Online]. Available: https://scipost.org/10.21468/SciPostPhys.3.3.021

[27] J. Preskill, "Quantum computing in the NISQ era and beyond," vol. 2, p. 79. [Online]. Available: https://quantum-journal.org/papers/q-2018-08-06-79/

[28] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," vol. 26, no. 5, pp. 1484–1509. [Online]. Available: http://epubs.siam.org/doi/10.1137/S0097539795293172

[29] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," vol. 23, no. 1, pp. 5–48. [Online]. Available: https://dl.acm.org/doi/10.1145/103162.103163

[30] Intel, "Export compliance metrics for intel® microprocessors." [Online]. Available: https://www.intel.com/content/www/us/en/support/articles/000005755/processors.html

[31] J. Wang, G. Guo, and Z. Shan, "SoK: Benchmarking the performance of a quantum computer," vol. 24, no. 10, p. 1467. [Online]. Available: https://www.mdpi.com/1099-4300/24/10/1467

[32] A. Wack, H. Paik, A. Javadi-Abhari, P. Jurcevic, I. Faro, J. M. Gambetta, and B. R. Johnson, "Quality, speed, and scale: three key attributes to measure the performance of near-term quantum computers," publisher: arXiv Version Number: 2. [Online]. Available: https://arxiv.org/abs/2110.14108

[33] Nvidia, "cuquantum accelerate quantum computing research." [Online]. Available: https://developer.nvidia.com/cuquantum-sdk

[34] J. Chow, O. Dial, and J. Gambetta, "Ibm quantum breaks the 100-qubit processor barrier," *IBM Research Blog*, vol. 2, 2021.

[35] G. García-Pérez, M. A. Rossi, and S. Maniscalco, "Ibm q experience as a versatile experimental testbed for simulating open quantum systems," *npj Quantum Information*, vol. 6, no. 1, p. 1, 2020.

[36] P. W. Shor, "Scheme for reducing decoherence in quantum computer memory," vol. 52, no. 4, pp. R2493–R2496, 2647 citations (Crossref) [2023-03-14]. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.52.R2493

[37] A. M. Steane, "Error correcting codes in quantum theory," vol. 77, no. 5, pp. 793–797, 1616 citations (Crossref) [2023-03-14]. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.77.793

APPENDIX A

Lists of Matrices

Equation A.1 shows the common Pauli matrices, X, Y, and Z. The X gate is analogous to a bit flip, the Z gate analogous to a phase flip, and the Y gate is a type of rotation gate around the Y axis of the Bloch sphere. Also shown here is the "bit and flip" gate, the XZ gate.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad XZ = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \tag{A.1}$$

Equation A.2 is the identity matrix. Any nxn matrix multiplied by its appropriate nxn identity matrix results in no change to the original matrix. The diagonal is all ones, and all other values are zeroes. For example, given an nxn matrix A, $AI = A$.

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{A.2}$$

Equation A.3 shows other matrices representative of common quantum gate operations. Of special import here is the Hadamard gate, H. The H gate is used extensively in quantum algorithms to leverage the inherent parallelism of qubits and their interactions with each other.

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \quad T = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{4}} \end{bmatrix} \quad SX = \frac{1}{2} \begin{bmatrix} 1+i & 1-i \\ 1-i & 1+i \end{bmatrix} \tag{A.3}$$

Equation A.4 illustrates different rotation gates. These kind almost be thought of as quaternion rotations.

$$R_x(\theta) = \begin{bmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix} \quad R_y(\theta) = \begin{bmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix} \quad R_z(\theta) = \begin{bmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{bmatrix} \tag{A.4}$$

Equation A.5 shows the "generic" R_ϕ gate.

$$R_\phi = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{bmatrix} \tag{A.5}$$

Echoed cross-resonance gate follows:

$$ECR_{q_0,q_1} = \frac{1}{\sqrt{2}}(IX - XY) = \frac{1}{\sqrt{2}} \begin{bmatrix} 0 & 1 & 0 & i \\ 1 & 0 & -i & 0 \\ 0 & i & 0 & 1 \\ -i & 0 & 1 & 0 \end{bmatrix} \tag{A.6}$$

Equation A.5 below shows simple two qubit states. Note the binary counting type of operations that are taking place here. A "cheater" way to help understand this is to look at the right-most qubit inside the Ket notation as the least-significant bit in the binary counting system. In the vertical matrix/vector representation the top-most element correlates to a zero value, and going down the rows in the matrix it is understood to increment the counting value by one. For example, $|11\rangle$ correlates to a 11_2 value, which in turn implies a 3_10 value.

Hence, the matrix/vector representation has a one value placed in the fourth column position (because these indexes start with zero).

$$|00\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad |01\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad |10\rangle = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad |11\rangle = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{A.7}$$

APPENDIX B

Nine-Qubit Shor Code

The Shor QEC code was introduced by Peter Shor in 1995 [36]. It spreads a single qubit's information across eight additional qubits, for a total of nine physical qubits. This means that forming a single logical qubit with this scheme requires nine physical qubits. This QEC scheme protects against one bit flip (Pauli-X) and/or one sign flip (Pauli-Z) error. These X and/or Z errors can occur on any of the physical qubits, and can also occur on the same qubit (which is logically equivalent to the third type of error, the bit and sign flip error together). The QCD is as follows.

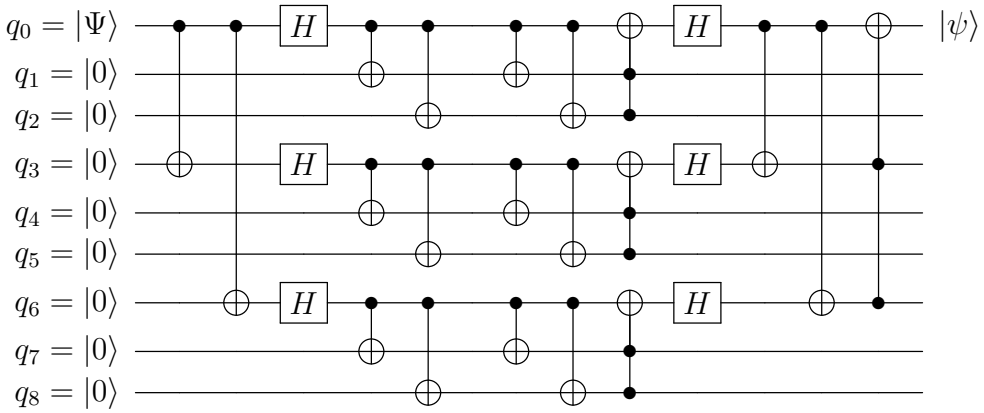

Figure B.1: QCD of the Shor QEC code.

The Shor code can correct arbitrary errors in the noisy channel.

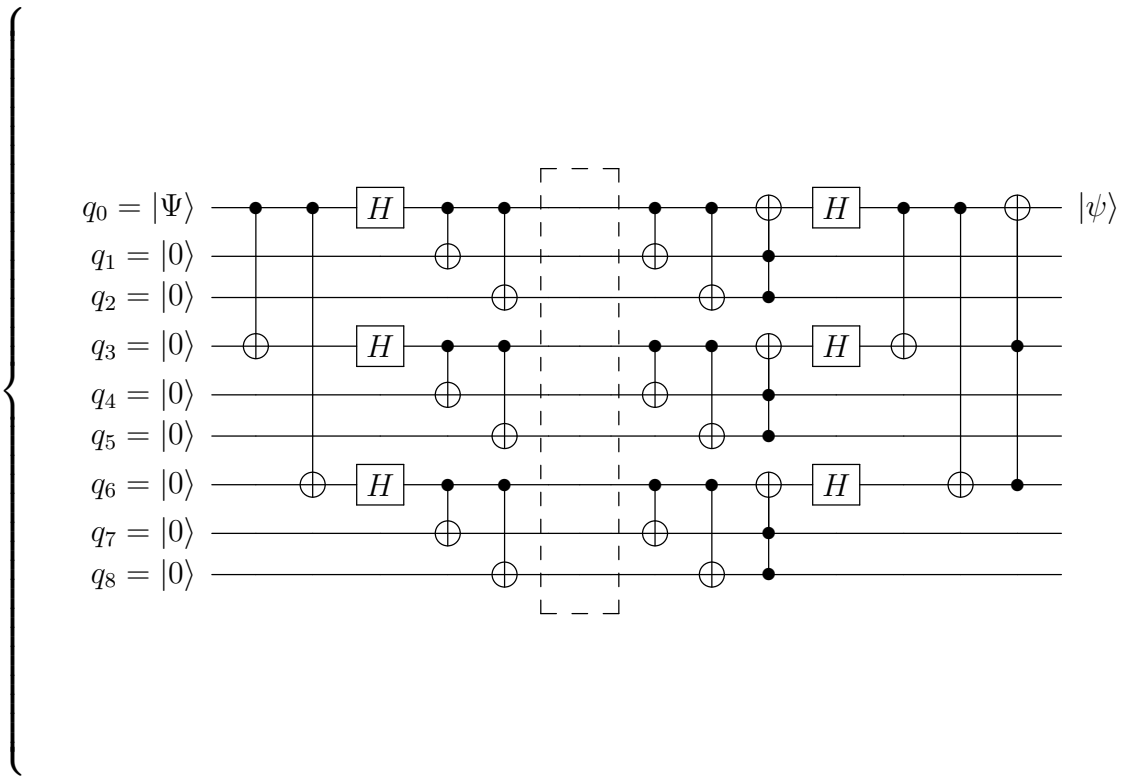

Figure B.2: QCD of the Shor QEC code.

APPENDIX C

Seven-Qubit Steane Code

The Steane QEC code was introduced by Andrew Steane in 1996 [37]. It spreads a single qubit's information across six additional qubits, for a total of seven physical qubits. This means that forming a single logical qubit with this scheme requires nine physical qubits. However, at a minimum an additional three extra "ancillae" qubits are needed as a type of scratchboard for error syndrome measurement. These ancillae qubits can be reused over and over again (after being reset to $|0\rangle$). Of course, in certain applications it may be better to use more than three ancillae qubits depending on how crucial it might be to minimize circuit depth by trading off additional qubits. This QEC scheme protects against one bit flip (Pauli-X) and/or one sign flip (Pauli-Z) error. These X and/or Z errors can occur on any of the physical qubits, and can also occur on the same qubit (which is logically equivalent to the third type of error, the bit and sign flip error together). The QCD is as follows.

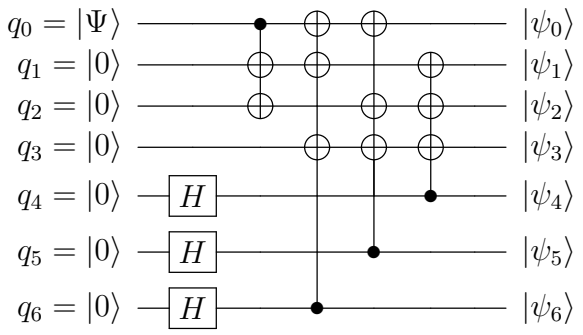

Figure C.1: QCD of the Steane encoding.

A portion of this research that is novel is a diagram showing the decoding step of the Steane code. The astute will observe it is simply the encoding operation reversed!

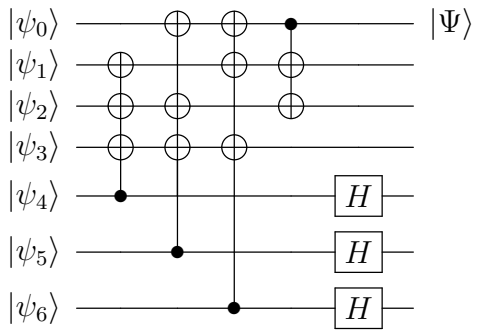

Figure C.2: QCD of the Steane decoding.

After the quantum state has been encoded onto seven physical qubits to form the logical qubit, you may use this logical qubit to perform certain transversal operations against other logical qubits that follow the Steane encoding scheme as well.

APPENDIX D

Methods: Campus Cluster and Nvidia A100 GPU's

Southern Illinois University Edwardsville (SIUE) participates in the Open Science Grid using a Slurm Compute Cluster. This hardware includes 10 CPU compute nodes each consisting of two AMD EPYC 7F52 16-core processors with 256 GB RAM and two 25 Gbps network interfaces. In addition to this CPU compute system also available are four GPU compute nodes each consisting of two Nvidia A100-PCIE-40GB graphics cards, two AMD EPYC 7502 32-core processors, 512 GB RAM, and two 25 Gbps network interfaces. Due to the GPU-bound nature of this project I asked to be granted access to the GPU nodes to perform some of my testing. After some back and forth I was eventually provisioned access to this system. With the Nvidia A100 GPU's at my disposal I was able to do scale testing far beyond what I would normally be able to accomplish with my home setup (See Methods: Apple Mac Pro A1289 and Nvidia T600 GPU's). The following are Python prerequisites I used to get the software to work on this system.

1. $ pip install –no-cache-dir cuquantum-python
2. $ pip install pytest
3. $ pip install qiskit
4. $ pip install qiskit[visualization]
5. $ pip install qiskit_algorithms

6. $ pip install qiskit-aer

7. $ pip install qiskit-aer-gpu-cu11

8. $ pip install qiskit-ibmq-provider

Below captures data related to GPU memory (VRAM) needed for qubit implementations with the current technology.

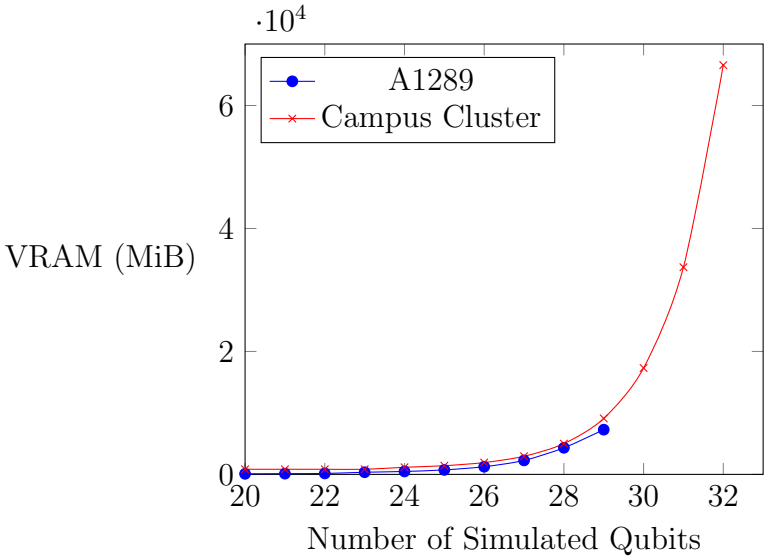

Figure D.1: GPU memory used while implementing simulated qubits.

APPENDIX E

Methods: Apple Mac Pro A1289 and Nvidia T600 GPU's

In May 2023, a free Apple Mac Pro A1289 was discovered left outside of Southern Illinois University Edwardsville's Science West building. This was picked up off the curb to be used for the GPU portion of this research. It was discovered that it had a faulty CPU tray, and it was replaced with a used part purchased from eBay for around $100 out of pocket. Upon the CPU replacement the device booted okay, but lacked a hard drive and an operating system. A SATA 1TB Solid State Drive (SSD) and tray were purchased for around another $75 out of pocket. Finally, two Nvidia T600 GPU's were obtained with money obtained from a Research Grants for Graduate Students program available through Southern Illinois University Edwardsville. These two separate lower-power GPU's were preferred over a single larger-powered GPU with the intent to see how they performed inter-communication across the bus.

Figure E.1: Apple Mac A1289 with Two Nvidia T600 GPU's.

APPENDIX F

Additional Information

Almost all code modules that were produced for this effort (except for some very early concepts) are provided at the below location. Additional code updates, and bug fixes are forthcoming, but generally the software is provided "as-is" without support.

`https://github.com/Shimmus/Entangled-Logical-Qubits/tree/main`

More information and contact information for the author can be found at the below locations.

`https://www.siue.edu/~dashimk/`

`https://davidshimkus.com/`