Due: June 8th, 2016
Date: June 7, 2016

Fault Tolerant Quantum Computing

Robert Rogers, Thomas Sylwester, Abe Pauls

---

1 Introduction

1.1 Project Summary

The goal of this project is to study the foundations, techniques, and current thought surrounding fault tolerant quantum computing. The subject encapsulates one of the greatest hurdles in creating a useful quantum computer. As such, a thorough understanding of fault tolerant techniques is critical to understanding any practical application (e.g. the goals of those trying to create quantum computers, the considerations of those designing procedures for use of and on quantum computers, etc.)

To carry out our study, we decided to review *Fault Tolerant Quantum Computation with Constant Error Rate* [1]. This, fairly lengthy, work lays the theoretical foundations for modern thought on the subject. It includes descriptions of quantum error correcting codes (QECCs) that work for qudits of prime dimension. These codes are: a generalization of CSS codes and a subset of those codes called polynomial codes. Using these codes, they proceed to generate fault-tolerant universal gate set (capable of describing any valid quantum circuit). They then provide several threshold theorems that show fault-tolerant quantum computation is possible so long as the error rate is below some constant value (the titular threshold). These results hold for a general qudit circuit using one of the above codes to compensate for a particular noise model. The types of errors considered vary depending on the section of the paper, but the main result holds for independent probabilistic noise. However, the most general threshold results hold for a "general noise" model that includes exponentially decaying correlations between errors, and the only restriction on the possible error operators is that they are within "some given distance" of the identity.

In this report we will summarize the general approach to fault tolerant quantum computing following the paper [1]. We will begin by introducing concepts central to quantum error correcting codes (QECCS), and present some particular codes of interest to the paper [1]. We will then discuss how a noisy quantum circuit is generally modeled. Using the concepts laid out in the first two sections we will discuss fault tolerant procedures including fault tolerant logic, zero-state preparation, and error detection/correction. Finally we present a sketch of the threshold theorem for polynomial codes and independent probabilistic noise [1].

2 Error Correction

2.1 Concepts, Classical Example, and Quantum Complications

The basic idea of error correcting codes is that, if we wish to store and transmit data without said data being compromised, we need to build some kind of larger mathematical structure to serve as a shield for structure containing the data itself. This shield must, to be effective, be able to both survive one or more errors (in the sense that accumulating $n \geq 1$ errors doesn't destroy the underlying data) and to be able to correct those errors over time, by restoring the original state.

One widely used classical error correction code is the repetition code. For example, we could replace every instance of the bit 0 in the data with the block 000, and similarly replace 1 with 111. This fulfills the first desired property of our shield, because the failure of a single bit in the code doesn't destroy the underlying

data (i.e. this code can correct for a 1 bit error). To see this, suppose that the probability of a particular bit flipping is p, and that we encounter the corrupted block 010. While we cannot say with 100% certainty that the block was initially 000 rather than 111, it is more likely; specifically, if the block was originally 000 then one bit flip has occured, while if the block was originally 111 then two bit flips have occured. This means the probability of 000 being correct is proportional to p, while the probability of 111 being correct is proportional to p^2, which is far smaller for systems with small values of p (for classical computers, p is often on the order of 10^{-17}- a truly miniscule ammount!). As for the second desired property of our shield, repairability, the classical repetition code is easy to fix: we simply flip the offending bit, restoring the block 010 to 000.

The quantum case, unfortunately, is not so simple. Firstly because a qubit can accumulate errors in its phase (i.e. $|0\rangle \rightarrow -|0\rangle$), a concept with no classical analog, and secondly because we cannot actually measure all of the qubits to see if and what errors have occured without destroying the state. This requires us to use a more clever combination of codes and corrections than before

The power of a quantum code lies in its ability to get around the measurements-collapse-states issue mentioned previously: while we cannot make a full measurement of the state (or for that matter many kinds of measurements), we CAN safely make CERTAIN KINDS of measurements to parse out the errors while leaving the information intact. This concept is key to the class of QECCs called Stabilizer codes (of which CSS codes and Polynomial Codes are an example).

2.2 Stabilizer Codes

Stabilizer codes are a more general class of QECCs subsuming CSS codes. In terms of our study, they are of interest mostly due to the power of the formalism used to describe them, and the intuition that formalism provides. The following discussion follows the discussion presented in *Nielsen and Chuang* [2].

Essentially, we want to construct codes in which special kinds of measurements can be used to figure out what errors have occured without damaging our information. It turns out that these measurements will be related to the group structure underlying the code we constructed. What kind of group structure we want built into our code will depend on what errors we're trying to correct; however, since any general, reversible error is reducible to a bit or a phase flip the choice of the underlying group stucture is obvious: the Pauli group.

To describe stabilizer codes, we're going to introduce a new formalism for thinking about error correction more suited to exposing and exploiting group structures. This formalism will include some description of the code(s), how unitary operations act on elements of the code(s), and how projective measurements act on elements of the code(s). Using those results, we must then provide a description of error detection and correction within the formalism.

2.2.1 Description of the code in Stabilizer formalism

Given a group, G, a stabilizer is a sub-group $S \subseteq G$, which fixes every element of some vector space V_S (i.e for all $|\psi\rangle \in V_S$ and $s \in S$, $s|\psi\rangle = |\psi\rangle$). Instead of concerning ourselves with V_S and the states therein directly, we will instead describe V_S via the generators of S. Hence an [n,k] stabilizer code consists of a vector space V_S stabilized by some $S =< g_1, ..., g_{n-k} >$ and a set of orthogonal spaces corresponding to the possible errors.

The group G of interest in our formalism is the Pauli group of rank n, G_n. This group is defined to be set of all n fold tensor products of Pauli matrices together with those related by multiplicative factors of ± 1, $\pm i$. This provides us a set of operators which acts on a vector space of size 2^n. Hence our code has the structure: some $V_S \subseteq \mathcal{C}^{2n}$ described by some set of $n-k$ generators and subspaces of C^{2n} orthogonal to V_S.

Note that for any given subgroup S, V_S might have a very trivial structure. Of course, such V_Ss would not be particularly useful as part of a QECCs. After some consideration it is possible to show that V_S is nontrivial iff $-I \notin S$, and the generators of S commute. Further, if these requirements are met $\|V_S\| = 2^k$. This essentially means that for some [n,k] stabilizer code we are encoding k qubits into an 2^n dimenional space where the valid codewords (where no errors have occured) occupy the subspace V_S of dimension

2^k and corrupted states occupy subspaces of the 2^n dimensional space orthogonal to V_S (the traditional interpretation of the [n,k] notation).

2.2.2 How unitary operators act in Stabilizer formalism

Normally, once we have our code we're interested in how unitary operations might act on its elements (e.g $U|\psi\rangle$) in order to capture the code's dynamics. Instead we're going to look at what unitary operations do to our list of stabilizers to accomplish the same task. For instance if we start with some V_S

$$s|\psi\rangle = |\psi\rangle \, \forall \, |\psi\rangle \in V_S \forall s \in S$$
$$\rightarrow U|\psi\rangle = Us|\psi\rangle = UsU^\dagger U|\psi\rangle \, \forall \, |\psi\rangle \in V_S \forall s \in S$$

We see that our new group of stabilizers is $USU^\dagger$.

For our formalism to remain convenient, we need the following property $USU^\dagger \subseteq G_n$, or, more generally $UG_nU^\dagger = G_n$. Otherwise, the new stabilizers might not have the simple algebraic properties associated with $g \in G_n$ which we'd like to exploit in our detection/correction schemes (described later).

The set $N(G_n) = \{U : UgU^\dagger \in G_n\}$ is called the normalizer of G_n.

2.2.3 How measurements are represented in Stabilizer formalism

Measurement in the computational basis may be described by observables corresponding to $g \in G_n$ in our formalism. Given some state $|\psi\rangle$ with stabilizer $< g_1, ..., g_n >$ we want to examine the measurement's effect on that stabilizer via changes to its generators. Note that the stabilizer $< g_1, ..., g_n >$ is particular to the state, but $< g_1, ..., g_{n-k} >= S$ are stabilizers for all states in V_S.

Since $g \in G_n$, g will either commute or anti-commute with elements of the stabilizer $< g_1, ..., g_n >$ for our state$|\psi\rangle$.

If g commutes with elements of the stabilizer then either g or $-g$ is an element of the stabilizer. Either way, the measurement will yield 1 with probability 1, and neither the state, nor the associated stabilizer will change.

If g anti-commutes with elements of S we follow a particular procedure. We organize $< g_1, ..., g_n >$ so the first m elements anti-commute with g. We can ignore all $m-1$ elements after g_1, since if some $\{g, g'\} = 0$ and $\{g, g''\} = 0$ then $[g, g'g''] = 0$. Then we replace g_1 with $\pm g$ depending on the result of the measurement (+ for a result of 1, - for a result of minus 1).

2.2.4 Description of error detection and correction in the Stabilizer formalism

First we will consider the sorts of errors we can expect in general, and whether or not we can correct those errors. We restrict ourselves to errors $E \in G_n$ (again, not very limiting, given that general error can be modeled as combinations of bit/phase flips).

If $E \in G_n$ then there are only a few ways in which it can have an effect on the generators of the stabilizer. It turns out, in short, that those errors correctable by our code are those errors

$\{E_i \in G_n : \forall E_i, E_j E_i^\dagger E_j \notin N(S) - S\}$

where N(S) is the normalizer of the stabilizer (defined the same way as it was above for G_n).

One can get a sense for why this is by examining the possible interactions between an error $E \in G_n$ and the generators of the stabilizer of $V(S)$. With some effort, one can see that if E anti-commutes with some generator of the stabilizer, we go to an orthogonal subspace of the code:

If the spaces weren't, then both E and g should stabilize some vector shared between the original subspace and the new subspace: $Eg|\psi\rangle = |\psi\rangle$. However, E and g should also anti-commute $Eg|\psi\rangle = -gE|\psi\rangle = -|\psi\rangle$. Meaning $|psi\rangle$ is necessarily zero, a contradiction.

If E commutes and has an effect on the state (i.e. E is not a stabilizer of $|\psi\rangle$), we cannot correct the error, since there is no guarantee of orthogonality. It's trivial to see that it is exactly these kinds of errors belong to $N(S) - S$ (those elements of G_n that commute with S, but are not part of S).

The following methodology can be used to detect and repair any (detectable) errors. Recall that all the generators of S, $g_1, ..., g_{n-k} \in G_n$ are, in fact, observables. We will simply measure each $g_1, ..., g_n$ in turn

producing a sequence of results. Due to the results of our above sections, this operation will not collapse the state, whether or not the state has been corrupted (consider the effect of measurement g_i on the stabilizer of the corrupted state $< E_j g_1 E_j^\dagger, ..., E_j g_n E_j^\dagger >$). If there was an error $E_j \in \{E_i\}$, we generate a "syndrome" some list of values, β_k, which are determined by our consideration of the action of a unitary operation. $E_j g_k E_j^\dagger = \beta_i g_i$. On detection of this error we simply apply the inverse $E_j^\dagger$ to correct.

The above assumes that the error has a unique syndrome. If it does not, it actually suffices to correct for one of the errors selected at random. This is due simply to the fact that if two errors have the same syndrome, then their composition $E_j^\dagger E_k \in S$.

It is also typical to define a distance akin to classical linear codes. Clasically we do this via the minimum number of places two codewords can differ. In our case this is the minimum weight of an error out of the class of correctable errors (number of non-identity contributions to the tensor product $E_j \in \{E_i\}$). It is possible to then show that a code of distance $d = 2t + 1, t \in N$ can correct up to t errors.

2.3 CSS Codes Generalized to F_p

While we generally treat quantum circuits and computational processes within the framework of the two dimensional computational basis, this framework is contingent on the computer being constructed from two-state quantum objects, qubits, such as isolated electron spins. In practice implementing a working physical quantum computer may be dependent on quantum systems that have more than two eigenstates, such as nuclear spins or hyperfine spin states. For this reason we would like to have a generalization for CSS codes that encompasses these higher level qubits. We define such quantum systems that have p eigenstates to be p-qudits. For these systems we define our CSS codes over the field F_p rather than F_2.

In considering error correcting codes for such systems we must first extend our definition for the types of errors that can occur. A qubit may undergo a binary bit-flip. The extension of the bit flip for a p-qudit is given by:

$$B\left|a\right\rangle = \left|((a+1) \bmod p)\right\rangle.$$

Successive applications of the generalized bit flip sequentially iterates the p-qudit through all of its eigenstates until it returns to it its original state. The p-qudit extension of the phase-flip error is given by:

$$P\left|a\right\rangle = \omega^a \left|a\right\rangle$$

It is instructive to consider the specific case of when $p = 2$. We can see that these generalized forms do in fact produce the familiar bit and phase-flip operators we are used to seeing in two state qubit systems.

In the same way that X, Z, and XZ can be used to construct the other Pauli matrices, we can also construct the set of higher dimensional Pauli matrix analogues through all possible compositions of B and P,

$$B^c P^{c'} \forall c, c' \in F_p.$$

These matrices form an orthonormal basis for the space of all matrices that act on a p-qudit, as do the m-fold tensor products of these matrices for all matrices that act on m p-qudits. Therefore, any unitary matrix operating on the m-tensor space of m p-qudits may be written as a linear combination of the set of these extended Pauli matrices.

With qubits we are able to correct spin flips by transforming the state into Fourier space via the Hardamard operation, correct a bit flip error and then transform back into the original space via a second application of the hardamard gate. This is possible because, for qubits, a phase-flip on a qubit appears as a bit-flip on the same qubit in Fourier space.

With the Fourier transform generalized for F_p

$$QFT_l\left|a\right\rangle = \frac{1}{\sqrt{p}} \sum_{b \in F} \omega^{lab} \left|b\right\rangle$$

we are similarly able to map bit-flip errors on a p-qudit to a phase-flip error and vice versa with the relations:

$$\forall c \in F_p, QFT_l P^c QFT_l^\dagger = B^{c/l}, QFT_l B^c QFT_l^\dagger = P^{cl}$$

where ω is the same phase used in the definition for the generalized phase-flip.

Just as a quantum error correction procedure which corrects errors that take the form of the Pauli matrices is able to correct a general error, it may be shown that under this generalization the same is true for a procedure which can correct the higher dimensional Pauli matrix analogues.

We may therefore use the stabilizer formalism to develop a CSS for p-qudits using these generalized "Pauli" errors to construct the necessary code-spaces.

2.4 Polynomial Codes over F_p

Polynomial codes are a small subset of of CSS codes which we define in the following way. First we define the set of all polynomials over p of degree at most d, $V_d = \{f(*) \in F_p[x] : deg(f) < d\}$. Then we choose m non-zero elements $\alpha_1, ..., \alpha_n \in F_p$. Finally we define the two classical codes which comprise the CSS code:

$$C_1 = \{(f(\alpha_1), ..., f(\alpha_m)) : f(*)\ inV_d\} \subset F_p^m$$
$$C_2 = \{(f(\alpha_1), ..., f(\alpha_m)) : f(*)\ inV_d, f(0) = 0\} \subset C_1$$

Where the encoding of state $|a\rangle$, $a \in F_p$ is

$$|S_a\rangle = \frac{1}{\sqrt{p^d}} \sum_{f \in V_d, f(0)=a} |f(\alpha_1), ..., f(\alpha_m)\rangle = \frac{1}{\sqrt{p^d}} \sum_{\omega \in C_2} |\omega + \vec{a}\rangle$$

When we write a vector in a ket we mean that each element of the vector is outer-producted with the other elements in order. For example, $|a, b, c\rangle = |a\rangle \otimes |b\rangle \otimes |c\rangle$.

Polynomial codes were of particular interest [1] because they allow for the implementation of fault tolerant logic in a particularly elegant way. Specifically, we will be exploiting the algebraic properties of the interpolation coefficients of a given polynomial.

3 Models of Error

In practice quantum computers are difficult to implement. The reason for this is that they require the qudits they are constructed from to be isolated from the environment and remain completely unaltered by any processes other than the logical gates which are applied to them. In reality such isolation is extremely difficult to achieve. Undesired interactions with the qudits surroundings, or even imperfect executions of quantum gates and measurements all introduce the possibility of error. These errors are by nature random, so in our modeling we consider faults to be probabilistic events that occur throughout the evolution of a quantum circuit. We will consider the ramifications of the error rates and circuit designs that will inform the basis for fault tolerant quantum computations.

3.1 Circuit Structure and Faults

A quantum circuit is constructed from some set of operational qudits and a time ordered sequence of quantum gates that perform operations on subsets of the qudits. We define the gates of a quantum circuit with t number of gates to be the set of operators g_i, where i runs from 1 to t. Using the density operator formalism we may write the final state output from a quantum circuit as,

$$Q \circ \rho = g_t \circ ... \circ g_3 \circ g_2 \circ g_1 \circ \rho$$

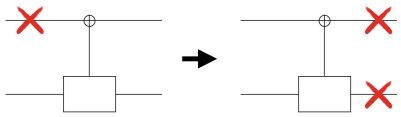

Figure 1: Example of error propogation though a CNOT

where ρ is the density matrix for the input state. The order of the composition of the gates are topologically determined by the structure of the circuit, with some variation allowed for gates that can be said to be implemented at the same time. The above description of a quantum circuit is without error. In order to introduce error we must introduce the notion of a location in a quantum circuit. A circuit may divided into some integer number of time steps, such that a given gate occurs at some time step t. Locations in a circuit are defined by the gates, rather by the time step at which the gate occurs, and the set of input qudits of the gate. We introduce error into this model by applying fault operators, $\mathcal{E}_i$, at locations in the circuit in between times steps. The resultant output of a circuit with a specific set of faults is then,

$$Q^{\mathcal{E}} \circ \rho = \mathcal{E}_t \circ g_t \circ ... \circ \mathcal{E}_2 \circ g_2 \mathcal{E}_1 \circ g_1 \circ \rho.$$

Each $\mathcal{E}_i$ represents the the composition of individual faults on each qudit. One specific set of $\mathcal{E}_i$, denoted by $\mathcal{F}$ is called a fault path. It is the list of specific faults that occurred in a single execution of the circuit.

To design a robust, fault tolerant procedure we need to consider all possible fault paths that can occur in that circuit. In general we assume that all faults have equal probability p, known as the error rate, of occurring. Hence larger fault paths are less likely to occur than smaller ones. In defining a general error rate of p we make the assumption that the faults are uncorrelated both in time and space. The output state of a noisy circuit is given by,

$$\left(\sum_{\mathcal{F}} \Pr(\mathcal{F}) Q^{\mathcal{E}}(\mathcal{F}) \right) \circ \rho$$

where $Pr(F)$ is the probability of the fault path $\mathcal{F}$ occurring.

It is important to distinguish the difference between faults and errors. A fault is a noise operation that occurs at a specific location within a circuit. The application of a fault on a quantum state in turn can produce a deviation of the state from the correct density operator. Whereas a fault is an event localized at a specific location in the circuit, an uncorrected error will persist throughout the evolution of the circuit and potentially propagate to other qudits.

3.2 Error Propagation and Spread

By using quantum error correction and encoding individual qudits into larger blocks of qudits we can make an operation robust to errors. However, even after implementing error correction there is yet another obstacle to overcome. Meaningful quantum computations necessitate logic gates that operate directly on the encoded states. By performing single quantum gates on the many encoding qudits simultaneously we open the door for error to propagate to other qudits in the circuit.

As an example consider an arbitrary control gate on two qudits. If an uncorrected fault occurred on the target qudit at some time before the control gate, after applying the control gate the error is still limited to the the target qudit. If instead the error had occurred on the control qudit (see Fig. 1), not only will the error persist on the control qudit but the target qudit will accumulate that error as well:

While error correction can remove errors it is not possible to perform error correction at every point along a circuit. For this reason, when constructing a quantum circuit we need to do so in a way that limits how "far" an error is allowed to propagate to other qudits.

When using encoded qudit blocks to correct for some number of errors, a single error no longer implies a failed logical operation. Even so, when implementing an encoded gate such as a single encoded control gate,

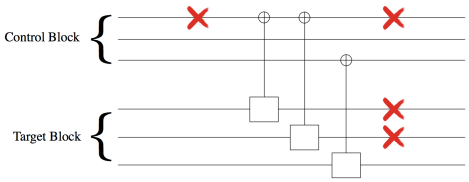

Figure 2: Example of a spread 2 circuit

attention needs to be paid to how the individual qudits in the control block are coupled via basic gates to the qudits of a target block.

As an example, consider the execution of an arbitrary control gate using a three qudit code that can correct for up to one error in the block. If we performed that operation using the encoded procedure shown in the Fig. 2, an error in the top qudit of the control block produces two errors in the target block, and the encoded target qudit is no longer recoverable. We define the extent to which errors in an encoded block will propagate to the other encoded blocks in an operation as the "spread".

More precisely, we say that a procedure which has some fault path with k faults, that operates on encoded blocks with no more than f qudits per block, using a code that can correct up to q errors, has a spread of l if $l(f + k) \leq q - l$. For a procedure to have a spread of l means that for each error that occurs in a particular input block for that operation, each of the output blocks from that operation will gain at most l errors. Therefore if we use a code that can correct at most one error in a block, we must use only procedures with a spread of 1 in order for it to function reliably.

4 Fault Tolerance

4.1 Introdution and Broad Concepts

Given the error model presented in the previous section, we need to consider what procedures and strategies are necessary in order to implement a quantum circuit fault tolerantly. By fault tolerant circuit we mean that given some error rate p for all its elements, the circuit may undergo some number of faults in its execution but still produce the correct output state that would have been produced had no faults occurred. A circuit being fault tolerant does not mean it is robust to any number of faults at any set of locations in the circuit. No matter what code or procedure used there will always be some fault path that corrupts the qudits to an extent that is unrecoverable. Instead the goal is to produce a circuit robust to some (physically reasonable) level of noise. This is accomplished by using fault tolerant procedures for state preparation and gates that limit the spread and the probability of error. By implementing a circuit fault tolerantly we are able improve from circuits that operate on single qudits in which each element has an error rate of p, to block encoded procedures with an overall fail rate of order p^2. As we will see in the discussion of the threshold theorem, by using multiple layers of encoding in which each layer utilizes fault tolerant procedures, it is possible to reduce the failure rate of a quantum circuit to an arbitrarily low level.

4.2 Fault Tolerant Logic

The goal of any program to produce fault tolerant logic gates is to fault tolerantly implement a universal set of gates. Essentially, the goal is to implement a set of gates that will allow for any arbitrary quantum computation. However, proofs of universality are rather complex and dense [1] and do not serve to illuminate the general design principles of fault tolerant logic. Instead, we will introduce those general design principles, and then provide an in-depth example of a particular gate which highlights the core concepts.

A general strategy for creating fault tolerant logic is to implement gates on encoded states in a *transversal* or *semi-transversal* fashion. A transversal implementation of a gate means the following. Suppose we have an n qudit gate acting on n encoded qudits. Let's label the dits of each encoded block 1 through m (m the

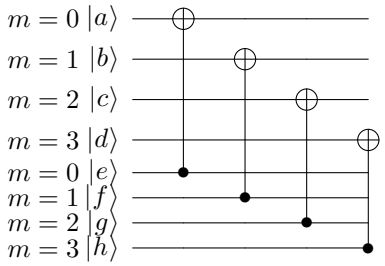

Figure 3: An example of transversal application of a CNOT between two encoded qudits.

length of the code). We apply the n qudit gate to each encoding dit labeled by the same label (see Fig. 3). The definition of a semi-transversal application is very similar, and contains only one salient difference. Instead of simply applying the gate on each of the qudits, we instead apply slightly different gates at each index. How the encoding gates are modified depends on what gate we're trying to encode.

The fault tolerance of this method of encoding gates is trivial. For an encoded n-qudit gate only n qudits will be involved in any encoding gate, and therefore the encoded gate has spread n (that is, one error in one of the qudit blocks can, at most, spread to n other qudits in the other qudit blocks).

The correctness of these methods, however, is less evident. That is, we do not yet know that the encoded gates acting on encoded states act in the correct fashion. Many cases are very easy to check, and the transversal/semitransversal implementations require very little modification.

In this section we will present one of the more challenging cases, the quantum fourier transform. It turns out that the QFT can be implemented semi-transversally using polynomial codes; however, there are some modifications to the strategy required to do this effectively. We follow [1].

Recall our definition for the fourier transform over the fieldF_p

$$QFT_l\,|a\rangle = \frac{1}{\sqrt{p}}\sum_{b\in F}\omega^{lab}\,|b\rangle$$

Consider the semi-transversal application of this operation to an encoded qudit:

$$\left|S_a^d\right\rangle = \frac{1}{\sqrt{p^d}}\sum_{f\in V_d, f(0)=a}|f(\alpha_1),...,f(\alpha_m)\rangle = \frac{1}{\sqrt{p^d}}\sum_{\omega\in C_2}|\omega+\vec{a}\rangle$$

Where d indicates the maximum degree allowed for the encoding polynomials.

$$QFT_{c_1}\otimes...\otimes QFT_{c_m}\,|S_a\rangle = \frac{1}{\sqrt{p^{d+m}}}\sum_{b_1,...,b_m\in F_p}\sum_{f\in V, f(0)=a}\omega^{\sum_{l=1}^m c_l f(\alpha_l)b_l}\,|b_1,...,b_m\rangle$$

We now define a set of unique polynomials s.t. $b(\alpha_l) = b_l$ with degree $deg(b) < m-d$. If we make this choice we see that the polynomial $f(x)b(x)$ is of degree $deg(f(x)b(x)) < m$ We choose $c_1,...,c_m$ as the interpolation coefficients of the $f(x)b(x)$. This means that $\sum_l^m c_l f(\alpha_l)f(\beta_l) = f(0)b(0)$

$$\rightarrow QFT_{c_1}\otimes...\otimes QFT_{c_m}\,|S_a\rangle = \frac{1}{\sqrt{p^{d+m}}}\sum_{b_1,...,b_m\in F_p, deg(b(x))<m-d}\sum_{f\in V, f(0)=a}\omega^{b(0)f(0)}\,|b_1,...,b_m\rangle =$$

$$\frac{1}{\sqrt{p^{m-d}}}\sum_{b_1,...,b_m\in F_p, deg(b(x))<m-d}\omega^{b(0)a}\,|b_1,...,b_m\rangle = \frac{1}{\sqrt{p^m}}\sum_{b\in F_p}\frac{\omega^{ab}}{\sqrt{p^{m-d}}}\sum_{b_1,...,b_m\in F_p, deg(b(x))<m-d, b(0)=b}|b_1,...,b_m\rangle$$

By considering the norm of the vector we can show that all bs in the second sum not represented by a zero of one of our selected polynomials vanish yielding

$$QFT_{c_1}\otimes...\otimes QFT_{c_m}\,|S_a\rangle = \frac{1}{\sqrt{p^m}}\sum_{b\in F_p}\omega^{ab}\left|S_b^{m-d-1}\right\rangle$$

Almost as desired. We'd like the state to be encoded in polynomials of degree d. Thankfully, this problem is easily remedied via a process called degree-reduction, which can be accomplished by quantum teleportation [1]. The degree reduction procedure is not shown here, for the sake of brevity.

This example serves to demonstrate the key concepts of fault tolerant logic and how polynomial codes might be exploited in implementing semi-transversal encodings of essential gates.

4.3 Fault Tolerant Zero-State Preparation

At some point in any consideration of fault-tolerant state preparation, we must assume that the procedures used to prepare prospective states are not fault tolerant. The general approach used to overcome the issues posed by this dilemma is as follows:

1. Assume that the majority of prepared states have no more than the number of errors that our error correcting codes can correct.
2. Define the sets T (set of all states with $< k$ errors) and F(set of all states with $\geq k$ errors).
3. Select 2 states at random from the pool of prospective states, $|\psi\rangle$ and $|\phi\rangle$.
4. Try to determine the relative position of $|\psi\rangle$ and $|\phi\rangle$ in T and F by detecting errors in a novel way.
5. Use whichever state $|\psi\rangle, |\phi\rangle \in T$

Note that the above procedure itself has to be resilient to a certain number of faults.

We present one implementation of such a procedure that is resilient to one fault [1]. Throughout this consideration we will assume that only one of the prospective states has any error associated with it. This algorithm utilizes a pool of 5 prospective encoded zero states $|\mathcal{S}_0\rangle$ prepared from states of the form $|0^m\rangle$ (m the length of the code). We will label each state $i = 1, 2, 3, 4, 5$. The first and fifth prospective states are our analogues of $|\phi\rangle$ and $|\psi\rangle$. Assume all logic gates applied are implemented via fault-tolerant(transversal/semi-transversal) methods.

We begin by addressing the possibility of dit-flips in state 1.

To do this we will apply a CNOT gate from state 1 to state 2. Then we perform a classical computation on the qudits of state 2. We copy each dit of state 2 m times, we then use these copies to perform m independent calculations, checking whether or not state 2 is of distance 1 away from C_2 or not. We record the results of these calculations in m "dit flip indication bits", let's call the string comprised of these bits β_1.

We would now like to address the possibility of phase flips in state 1. However, there is the danger of dit flips having occurred in state 3. If we apply the QFT and then the CNOT gates, we run the risk of errors back-propagating and fully corrupting state 1. We solve this problem via the following.

We will do the same procedure as above between states 3 and 4 to produce another m "dit flip indication bits", let's call the string comprised of these bits β_3. We will then apply a QFT to state 1 and 3, and conditionally apply a CNOT gate from from 1 to 3 depending the ith entry of β_3. This may also be described as the application of a generalized Toffoli applied between the ith qudits of states 1, 3, and ith entry of β_3. The same classical computation we did for dit flips follows, using the dits of state 3. This produces m "phase flip indication bits", ξ_1. We complete the phase flip detection by applying $QFT^\dagger$ to state 1.

Finally, we compile the results of the testing to determine if state 1 belongs to T or F. We do this by checking if there exists one bit location for β_1 and ξ_1 that, if changed, will take β_1 into C_2 and ξ_1 into C_1 (of the dual code). If so we use state 5 since state 1 belongs to F, and if not we use state 1.

Now, let's try to make sense of this procedure. The first two parts of this algorithm make intuitive sense. Essentially we're compiling information about possible dit and phase flips in state 1. Why the final part of the algorithm works is less obvious, and the underlying reason is beyond the scope of this report.

We conclude this section with some comments on the above procedure, and fault tolerant zero-state preparation in general. Note that the error detection in this procedure was all done classically, and hence would not be applicable if we wanted to use it to do error correction in the circuit we're using for the computation. Further, there is a large overhead in the number of states required to produce a single usable state. This

quality of our procedure reflect general trends in fault tolerant zero-state preparation procedures[3]. Finally, recall that this particular algorithm is only robust up to a single fault; however, it can be concatenated with itself to form algorithms that are robust up to the desired number of faults [1].

With these tools in hand, we can now describe how to perform, fault tolerantly, the most critical procedure relevant to reliable quantum computing.

4.4 Fault Tolerant Error Correction

Now that we've come to the most important fault tolerant procedure, ironically, there isn't much that needs to be said. The considerable legwork in developing useful QECCs, fault tolerant logic, and fault tolerant zero-state preparation leaves the procedure remarkably easy to describe.

First, we use fault tolerant logic to copy the error syndrome from the block to the ancillary zero-state (which has been prepared fault tolerantly). To do this we define the following unitary operation utilizing M_i the measurement operators associated with our fundamental errors for the field F_p:

$U\left|S_a\right\rangle\left|S_0\right\rangle = \sum_i (U_i M_i \left|S_a\right\rangle)\left|S_i\right\rangle$

It's easy to see that this operation preserves inner products in the QECC since [2]:

$\left\langle S_a\right| \left\langle 0|U^\dagger U|S_b\right\rangle |0\rangle = \sum_{ij} \left\langle S_a|M_i^\dagger M_j|S_b\right\rangle \delta_{ij} = \sum_i \left\langle S_a|M_i^\dagger M_j|S_b\right\rangle = \langle S_a|S_b\rangle$

We then use fault tolerant logic to correct any errors that have occurred by using controlled gates which correspond to our fundamental errors for F_p.

The advantage of this sort of procedure is that it does not require intermediate measurements which can be technically complicated to implement. It does, however, require a high overhead of fault tolerantly prepared zero-states (a procedure that already has an intrinsically high overhead).

5 Threshold Theorem

Even the cleverest of error correcting codes are subject to damage from the random fluctuations resulting from the quantum nature of any real-world circuit we could attempt to physically build. In this paper, we have considered a few particularly clever codes, and the different ways they can be used to make procedures more resistant to such fluctuations and faults; it remains, however, to be shown that such strategies can actually be relied upon to work. Intuitively, we need to show that there is some benchmark such that, if we reach it, our computations can be considered reliable, and that achieving circuits which meet said benchmark are physically realizable with an acceptable (i.e. sub-exponential) amount of effort. The precise statement of the theorem is as follows:

> The Threshold theorem for probabilistic noise [1]:
>
> Let $\epsilon > 0$. Let C be a computation code using a set of gates G. There exists a constant threshold $\eta_c > 0$ and constants c_1, c_2, c_3 such that the following holds. Let Q be a quantum circuit with n input qudits (qudits), which operates in t time steps, uses s gates from G, and has v locations. There exists a quantum circuit Q' which operates on $n * O\left(log^{c_1}\left(\frac{v}{\epsilon}\right)\right)$ qudits (qudits), for time $t * O\left(log^{c_2}\left(\frac{v}{\epsilon}\right)\right)$, and uses $v * O\left(log^{c_3}\left(\frac{v}{\epsilon}\right)\right)$ gates from G such that, in the presence of probabilistic noise with error rate $\eta < \eta_c$, Q' computes a function which is within ϵ total variation distance to that computed by Q.

In essence, the theorem states that, given any desired accuracy for the final result, we can achieve that accuracy simply by keeping the noise in our circuit below some particular constant threshold. Modern circuits aren't there yet, in the sense that the best noise level we've been able to manage is still a few orders of magnitude worse than the theoretical thresholds we need to beat, but we nonetheless have the important fact that we can, in principle, create reliable circuits.The proof of the theorem is rather mathematically dense, so in this paper we seek only to illustrate the general idea of the proof [1, 2].

The whole proof hinges on the idea of recursive encoding, i.e. simulating a qudit in a code C_k with a block of qudits in a code C_{k+1}, and iterating until the errors have been brought down to an acceptable level. Mathematically, we can see that if the failure probability in the original circuit C_0 was $p_0 = p$, and if each

single qudit in C_0 is encoded by a block $2m-1$ qudits in C_1 that, due to majority voting, we would need m failures to occur at the C_1 level in order to cause an overall failure. The probability of failure would thus be proportional to the m^{th} power of p, namely cp^m. If we then performed an additional layer of coding, taking each logical qudit in C_1 into a $(2m-1)$-block in C_2, the error rate would thus be $c(cp^m)^m = c^{m+1}p^{m^2}$; next, for C_3, we'd get $c^{m^2+m+1}p^{m^3}$, and so on. Inductively, we see that the code C_k should have a failure rate $c^{m^{k-1}+m^{k-2}+\ldots+1}p^{m^k} = c^{\frac{m^k-1}{m-1}}p^{m^k} = c^{\frac{-1}{m-1}}\left(c^{\frac{1}{m-1}}p\right)^{m^k}$.This means that, for $p < c^{\frac{1}{m-1}}$, the failure rate decreases super-exponentially with k, meaning we have an error threshold below which recursive encoding can give us arbitrarily low effective failure rates!

We also need to show that this can be done with a reasonable amount of effort; the super-exponential dependence on k implies that k shouldn't have to get that large for the circuit to work for us, but it would still be nice to have some estimate for it. Let d be a constant representing the "size"of the original circuit, in the sense that d is an upper bound for the number of operations that the circuit will do. When we go from C_0 to C_1 we're replacing each one qudit with a block of $2m-1$ qudits, effectively increasing the size of the circuit to dm.

Inductively, we see that the size of C_k should be $d(2m-1)^k$, which is merely exponential in k. This means that the size of the simulating circuit is merely a poly-logarithmic function of the desired error rate, as the theorem itself stated.

6 Conclusion

Given the threshold result, it becomes apparent that errors are not, in principle, a barrier to reliable quantum computation. It also provides a concrete goal in the design of quantum computers: it specifies exactly how noisy our system can be. In modern research, there have been many improvements on the theoretical threshold presented in this paper (even at the time the result, although the most general, was not the best threshold available). Nevertheless, a thorough comprehension of the concepts aids us in understanding a great deal of the literature on quantum computers.

References

[1] Aharonow, D. and Ben-Or, M. Fault Tolerant Quantum Computation with Constant Error Rate. *SIAM Journal on Computing* 38 no. 4 (December 2008): 1207-1282.

[2] Nielsen, M. and Chuang, I. Quantum Computation and Quantum Information: 10th Anniversary Edition. Cambridge: Cambridge University Press, 2010.

[3] Cross, Andrew. Fault-tolerant quantum computer architecture using hierarchies of quantum error-correcting codes. Cambridge: MIT Department of Electrical Engineering and Computer Science, 2008.