Error Aware Clifford Circuit Compilation

Alan Robertson

A thesis submitted in fulfilment of the requirements for the degree of

Master of Philosophy

School of Physics
University of Sydney

2019

Acknowledgements

With thanks to Steven Flammia for supervising and supporting this project, Robin Harper for useful discussions and Isabel Colman for copy editing.

Abstract

We demonstrate that small quantum memories, realised via quantum error correction in multi-qubit devices and implemented using noisy Clifford circuits, can benefit significantly from tailoring the choice of code to the error model of the Cliffords gates and environment. We present a simulation of these errors and the results of searching across random quantum error correcting codes and implementing the associated Clifford encoding circuits. This modeling incorporates a better representation of the experimental complexity involved in implementing these codes and demonstrates that tailored codes can outperform the Steane code with more realistic experimental noise. Depending on the error model, they are required to surpass the fault tolerant threshold. These Pauli error models correspond to individual Clifford generators and environmental noise on the wires. We present error models which incorporate independent identically distributed (IID) errors and biased Pauli errors in the construction and optimisation of Clifford circuits associated with maintaining a quantum memory.

Contents

Chapter 1

Introduction

Despite affording the most powerful applications in large scale problem solving to date, semiconductor transistor based computation is rapidly approaching a threshold where quantum effects compromise the fidelity of computations [74]. The exponential growth of classical computational power has been driven by increases in transistor counts, an associated decrease in transistor size and downscaling of integrated circuits. Unfortunately this growth is bounded; for transistors of size 7nm or smaller we observe quantum effects that cause information loss and unpredictable behaviour in microprocessors. Attempts to further increase the computational power of classical devices solely by reducing the scale of the system requires either the suppression of quantum effects or the replacement of classical computation with quantum information processing [28].

Beyond scaling concerns, quantum devices can perform computations in a manner that is believed to be more efficient than a classical Turing machine. A number of quantum algorithms with definite asymptotic improvements over their classical counterparts have been devised [23, 64, 37]. These algorithms more efficiently solve such problems as integer factorisation, systems of linear equations and the simulation of nano-scale physical systems [44, 23].

As the development of quantum systems trends away from early 'proof of concept' devices to noisy, intermediate-scale quantum (NISQ) systems comprised of 50-100 qubits, we may begin to see quantum circuits that perform tasks surpassing the capabilities of classical computers[58, 20, 44], however, noise severely limits the size of the circuits that can be executed reliably. In order to realise the advantages of NISQ devices we must find methods of reducing the error rate of quantum circuits[34]. Classical computational devices have often turned to error correction in the face of noisy environments, or high physical error rates; examples of this include error correcting memory and the use of redundancy checks, cryptographic hashes and Reed-Solomon codes for error detection and correction of hard drive memory.

It is expected that in order to reliably store information and perform operations scalably on NISQ devices, some form of error correction will be required [16, 17, 49, 51]. It has been previously shown that gains can be made in error correction schemes by tailoring the choice of code and decoder to the error model[68, 63, 29]. These gains in the logical error rate of small quantum memories translate into larger reductions in the overhead required to perform fault tolerant computation[35, 50].

These gains are typically made using some additional information about the error channel to provide a more accurate, but still approximate error model [19]. This characterisation of quantum systems is typically performed by quantum tomography methods. Despite the fact that these methods completely characterise the state or operation, they scale poorly with the size of the system and become prohibitively resource intensive [67]. Recent advancements have instead focused on using randomised benchmarking methods to characterise the dominant sources

of error in a system, and provide an associated error model [6, 13, 26].

Very recently, inroads have been made into using heuristic approaches to error aware quantum compilation [54]. This complements higher level approaches to reducing the error rate of NISQ systems as it tethers the architecture of the system to particular noise and error reduction approaches.

In order to achieve the degree of fidelity required for quantum computation, we must have methods that provide an accurate determination of the state of a quantum system and the characterisation of the operations that act on it, including these quantum gates and environmental losses.

Previous work[71, 10, 63] has demonstrated optimising decoders and codes for a given error model, or for a given code optimising the compilation of the associated Clifford circuit for encoding, decoding or measurement. As a combination of these approaches for a given error model we can consider optimising the choice of code based on the error rate of the implemented Clifford circuits.

As recent developments in quantum devices now have a sufficient number of qubits to implement these circuits, we require better device agnostic quantum compilation techniques in order to approach the fault tolerant threshold. The errors currently plaguing these devices are poorly represented by approximations of a general environmental noise, and are often correlated across qubits or strongly biased towards particular classes of errors[53, 6, 4]. Some devices demonstrate error channels with physical error rates that drift depending on the time since the device was last calibrated[39]. Compiling circuits that better tolerate these errors while encapsulating the complexity associated with the particular error model is a potential component in any approach to developing 'full stack' quantum computation [3].

By simulating the errors acting on the associated circuit we find that significant advantages can be found in optimising the choice of code for a given error model associated with a particular device. In particular we demonstrate that the comparative performance of codes under naïve error schemes does not translate to the same comparative performance when circuit noise is simulated.

Classical systems are built upon a stack of prior dependencies and systems that provide opaque interfaces to the calling program. Each command in a high level programming language sees itself compiled into a set of commands in a lower level language. This process is repeated with the lower level languages until a set of instructions is produced that can be understood by the central processing unit of the computer.

Modern programming languages often implement options for compiler heavy optimisation that provide performance improvements by using architecture aware modifications to the compilation pipeline[3]. This architecture aware optimisation is typically far more resource intensive than the typical replacement strategies that could be used[70, 62]. If a program is to be run many times, then if the optimisation is successful and the run time of the resulting program has been reduced, then the time lost to the build process may be quickly recouped. Unfortunately, this optimisation involves knowledge of the quirks and advantages of particular physical architectures; and other information obtained via diagnostics, profiling and characterisation.

A similar analogy can be drawn to the trend in NISQ systems; we have begun to see the abstraction of the physical control of the devices behind intermediate representations such as QASM [18, 46]. Unfortunately, at this point the analogy ends. While classical computers can efficiently simulate other classical computers; there exist several no-go theorems that indicate that it is not possible to efficiently simulate some classes of quantum operations using a classical

Turing machine[1].

As a compromise between efficient classical computability and universal computation, we chose to provide a proof of concept by considering only sets of quantum operations that are classically efficiently simulable. In this case, the Pauli and Clifford operators [1].

Chapter 2

Quantum Computation, Noise and Error Correction

In this chapter we will discuss quantum noise with a particular focus on Pauli noise and stabiliser based quantum error correction.

We will begin with a discussion of a general approach to universal quantum computation using a restricted gate set. As modern classical systems use small sets of gates termed assembly instructions to implement more complex operations, it has been proposed that an equivalent quantum assembly language could in turn be constructed from a universal quantum gate set. The foremost difficulty with this approach is the 'efficient' decomposition of an arbitrary quantum operation into elements of the gate set, or rather, instructions in the quantum assembly language.

'Efficiency' is a somewhat open ended goal, and again, with multiple relevant figures of merit. Classically we can consider reducing the number of operations required to implement a particular gate (often by increasing the number of elements in the gate set), reducing the overall runtime of the program or reducing the number of elements in the gate set and hence the implementation difficulty. For our purposes, we will consider our figure of merit to be the error rate of the implementation of the operation.

Within our 'quantum computer' states can be considered analogous to registers. These probabilistic registers are highly sensitive to errors; a small error acting on the system gives a non-zero probability of obtaining the wrong measurement result. We can decompose the error map acting onto the state into a finite set of possible errors and the probability with which they occur. For the Pauli basis, the number of possible errors grows with the number of qubits in the register n as 2^{2n}. This can be compared to the set of possible bit flip errors acting on a classical system, that grows as 2^n.

With a larger space of possible errors, and a much greater sensitivity to these errors due to the probabilistic nature of the register; it becomes a necessary to implement methods to suppress the overall error rate and protect the information stored within our register.

2.1 The Heisenberg Representation of Quantum Mechanics

Suppose we have a register of a quantum computer with state $|\psi\rangle$, to this we apply some unitary operations A and B, then

$$BA\,|\psi\rangle = BAB^\dagger B\,|\psi\rangle\,. \tag{2.1}$$

We can see that applying B transforms the operation A under conjugation

$$A \to BAB^\dagger. \tag{2.2}$$

The operator so described will still act on $|\psi\rangle$ in the same manner as BA did, but we can consider now how operations acting on a quantum state evolve independently of the state vector. As this evolution is linear, we can reconstruct any state vector by tracking how this final operator acts over a set of matrices that span the register space.

At this point, it seems prudent to introduce a couple of groups. The first of these are the Pauli matrices, which are the unitary infinitesimal generators of SU(2) and are comprised of the identity matrix and the following three matrices:

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{2.3}$$

Each pair of these Pauli matrices anti-commute unless they are identical or one is the Identity. The Pauli operators are then the group of all tensor products of n Pauli matrices. This tensor product of Pauli generators is also termed a Pauli string. It is worth noting that we will use the notation X, Y, Z to refer to the operators $\sigma_x, \sigma_y, \sigma_z$. We define the Pauli group $\mathcal{P}^n$ to be generated by all single qubit Paulis acting over n qubits.

The urgency of introducing this group is that the Pauli group spans our register space. As $\sigma_y = i\sigma_x\sigma_z$ the evolution of the state can then be tracked by calculating the evolution of these $2n$ single qubit operators [36].

Another group that it will be useful to define are the Clifford operators. The Cliffords are a subgroup of the normaliser of the Pauli operations and are defined as the set of unitaries $\mathcal{C} : \mathcal{P}^n/\pm I \to \mathcal{P}^n/\pm I$ under conjugation. That is to say that the Cliffords act as a map from Paulis to Paulis. While the action of an arbitrary unitary operation could take our initial Pauli operator to just about any unitary operator our generating set grows from $2n$ to the set of all completely positive, trace non-decreasing unitary matrices. To sidestep the scaling problems associated with an arbitrary unitary operation we can restrict the actions on our register $|\psi\rangle$ to only those that map elements of the Pauli group to each other. This constraint should maintain the size of our spanning set.

The full set of Clifford operations can be generated from just the Hadamard, CNOT and Phase gates which are defined as

$$\text{Hadamard} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \text{Phase} = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}, \text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{2.4}$$

The last of these Clifford operations, the CNOT gate acts on two qubits, the first is the *control* qubit while the second is the *target*.

As the evolution of the operator acting on the register is now constrained to only maps between Paulis, we simply calculate the overall map acting on each element of the spanning set. This map will itself, necessarily be an element of the Clifford group. Armed with an efficient representation of the evolution of a state as operators are applied to it, we can now begin to consider how this representation can be used in the context of quantum computation.

2.2 Quantum Computation

A state undergoing some controlled change can be described as an act of computation.

Classical computation is defined in terms of functions f that act on binary vectors in a canonically defined boolean algebra $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$. Each of these functions can be described in terms of a series of unary and binary boolean operators. There exist a subset of boolean operators which can realise all other boolean operators using only elements of this subset. These are the NAND and NOR gates. Some circuit comprising only of these gates can exactly represent the action of any other boolean gate, hence these are termed 'universal' gates. As our computation operations can be decomposed into these boolean operators any classical computation can be described solely in terms of universal gates.

From an hardware perspective, we can implement our function using nothing but some universal gate set. This technique permits the representation of the set of all functions that act on binary vectors in terms of a small number of gates. Rather than requiring specialised hardware for each function, we can compile the function as a sequence of gates acting on different bits in the register.

From this well trodden classical framework, we can draw comparisons to quantum computation.

Quantum operations take the form of completely positive, trace non-increasing operations. We will define quantum gates to be a subset of operations that are trace preserving unitary operations acting on a Hilbert space. As these gates are unitary they must satisfy the property $UU^\dagger = I$ and hence all quantum gates must be reversible. Each of these gates can be described in terms of a series of unary and binary gates. There exists a subset of quantum operators[22] that may realise all other quantum operators using only elements of this subset. These are universal quantum gates, and are a direct analogy to our classically universal gates.

Unfortunately, as quantum gates must be reversible, the NAND and NOR gates cannot be used as the basis of universal quantum computation. Instead, an example of a universal quantum gate is the Toffoli gate, and any quantum operation can be described solely in terms of almost any choice of Toffoli gate. The gate itself is a controlled unitary operation targeting a single qubit. Any 2×2 unitary matrix U gate may be expressed as

$$U = \begin{pmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{pmatrix} = \begin{pmatrix} e^{i\delta} & 0 \\ 0 & e^{i\delta} \end{pmatrix} \begin{pmatrix} e^{\frac{i\alpha}{2}} & 0 \\ 0 & e^{\frac{-i\alpha}{2}} \end{pmatrix} \begin{pmatrix} \cos(\theta/2) & \sin(\theta/2) \\ -\sin(\theta/2) & \cos(\theta/2) \end{pmatrix} \begin{pmatrix} e^{\frac{i\beta}{2}} & 0 \\ 0 & e^{\frac{-i\beta}{2}} \end{pmatrix}. \tag{2.5}$$

Where α, β, δ and θ are real valued. For a special unitary matrix, $\det(U) = 1$ and hence $e^{i\delta} = \pm 1$. We can then ignore δ for special unitary matrices as it can be folded into the second matrix. This decomposition of U can be described in terms of their actions on the qubit.

$$P(\delta) = \begin{pmatrix} e^{i\delta} & 0 \\ 0 & e^{i\delta} \end{pmatrix} \tag{2.6}$$

acts as a phase shift.

$$R_z(\alpha) = \begin{pmatrix} e^{\frac{i\alpha}{2}} & 0 \\ 0 & e^{\frac{-i\alpha}{2}} \end{pmatrix} \tag{2.7}$$

acts as a rotation by α about $\hat{z}$. The last matrix in the sequence is another application of R_z.

$$R_y(\theta) = \begin{pmatrix} \cos(\theta/2) & \sin(\theta/2) \\ -\sin(\theta/2) & \cos(\theta/2) \end{pmatrix} \tag{2.8}$$

acts as a rotation by θ about $\hat{y}$.

One final gate that we will briefly encounter is the T gate.

$$\mathrm{T} = \begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{4}} \end{pmatrix} \tag{2.9}$$

The T gate is the square root of the Phase gate, and is the first gate that is not efficiently classical simulable that we have encountered.

We can now attempt to compile a general unitary operation in terms of $\wedge_m(U)$ gates, which are in turn decomposed into some unitary gate set. We can define a new gate where U is applied to a qubit in the register conditioned on the state of m other qubits. This controlled U gate $\wedge_m(U)$ acts as

$$|\psi, k\rangle = \begin{cases} u_{k0}\,|\psi, 0\rangle + u_{k1}\,|\psi, 1\rangle & \text{if } \psi_i = 1 \forall i \in m \\ |\psi\rangle & \text{otherwise.} \end{cases}$$

The matrix corresponding to this function is given by

$$\wedge_m(U) = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \ddots & & \vdots & \vdots \\ \vdots & \vdots & & 1 & 0 & 0 \\ 0 & 0 & \cdots & 0 & u_{00} & u_{01} \\ 0 & 0 & \cdots & 0 & u_{10} & u_{11} \end{pmatrix}. \tag{2.10}$$

It can be seen for the the two qubit case where $U = \sigma_x$ that $\wedge(\sigma_x)$ is the CNOT gate.

"Almost any" choice of $\wedge_2(U)$ satisfies the conditions of our generalised Toffoli gate, and can be used as an elementary gate for universal quantum computing by applying the gate different pairs of qubits in the register. That is to say, that any completely positive, trace preserving map can be decomposed into a series of $\wedge(U)$ gates [2].

In the most general form, a gate $\wedge_1(W)$ can be constructed if there exists some set of gates V_1, V_2, V_3 such that $V_1\sigma_x V_2 \sigma_x V_3 = W$ and $V_1 V_2 V_3 = I$. It has been shown that such a construction always exists if W is a special unitary matrix [7].

We will adopt a notation of 'quantum circuits' where qubits are represented as wires and gates act on these wires. Throughout this thesis, circuits will read left to right. Controlled gates will be indicated by verticle lines with control qubits indicated by filled circles. The set of controlled $\wedge_n(X)$ gates are denoted by the $\oplus$ symbol, as shown in figure 2.1.

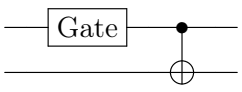

Figure 2.1: A CNOT gate

The example circuit above contains two qubits with a gate acting on the first and a $\wedge_1(X)$ gate acting on the second with the first as a control. Using this notation figure 2.2 shows we can now represent a decomposition of a $\wedge_1(W)$ gate.

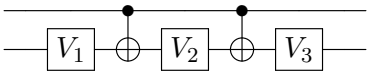

Figure 2.2: Circuit for the $\wedge_2 W$ gate.

If the control bit is 0 then the operation applied to the target qubit is $V_1 V_2 V_3$, which is an identity operation and leaves the target qubit unchanged. Conversely if the control bit is 1, the operation is $V_1 \sigma_x V_2 \sigma_x V_3$ and the W gate is applied to the target.

This methodology can be extended to three qubits to construct $\wedge_3(W)$. Let V be a unitary matrix such that $V^2 = W$. We can compile $\wedge_1(V)$ and $\wedge_1(V^\dagger)$ from the method presented above. Figure 2.3 shows this $\wedge_3(W)$ gate.

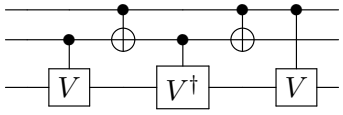

Figure 2.3: Circuit for the $\wedge_3 W$ gate.

From this circuit we can construct a truth table of the operations on the target qubit from the basis states of the two control qubits.

$\wedge_2 W$	0	1
0	I	$VV^\dagger$
1	$V^\dagger V$	VV

Table 2.1: Truth table for the $\wedge_2 W$ gate, as given by the circuit 2.2. As $W = V^2$, this table reduces to $\wedge_2 W$

This method can be expanded to implement any $\wedge_n(U)$ gate, and the $\wedge_n(U)$ gate can then be used for universal quantum computation [9]. As the $\wedge_1(V)$ gate can itself be constructed from some set of gates V_1, V_2, V_3 and CNOT, these form a universal gate set. The minimum requirement to perform universal quantum computation is then the ability to implement some set of V_1, V_2, V_3, their inverses (for $\wedge_2(V^\dagger)$) and the CNOT gate across any pair of qubits in the register.

The problem of quantum computation is then reduced to decomposing a unitary operation in terms of a particular choice of $\wedge_n(U)$.

This does not answer any questions about the efficiency of a particular decomposition (as can be heuristically determined by various metrics such as gate counts), or a particular choice of $\wedge(U)$, but does provide a working model.

2.3 Clifford Circuit Computation

The previous section detailed the decomposition of a unitary operator into a set of elementary gate operations. This process is the *compilation* of the quantum operation into a set of standard gates. The benefit of this compilation scheme is that a quantum device need only implement the these universal elementary operations and using these in sequence rather than physically tuning the device to implement each unitary gate.

While the previous section has developed a framework for constructing the circuits required for universal quantum computation, here we will limit ourselves to Clifford circuits. While Clifford gates are not themselves universal[1], we can still consider a rich environment of compiler optimisation problems with a limited gate set.

In addition to $\wedge_1(\sigma_X)$, from a special case of the circuit 2.2 we can construct $\wedge_2(\sigma_Z)$ and $\wedge_2(\sigma_Y)$, as shown in figure 2.4.

$$\wedge(\sigma_Z) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \qquad = \quad Z \quad = \quad H \quad H$$

Figure 2.4: A controlled Z gate constructed from Clifford gates.

Similarly figure 2.5 shows we can construct a controlled Y operation from the Clifford generators.

$$\wedge(\sigma_Y) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -i \\ 0 & 0 & i & 0 \end{pmatrix} \qquad = \quad Y \quad = \quad P \quad P \quad P \quad P$$

Figure 2.5: A controlled Y gate constructed from Clifford gates.

Unfortunately, as the Clifford gates themselves are not universal, they cannot implement any universal gates. Our $\wedge(U)$ gates cannot be constructed without also including at least one 'truely quantum' gate [9]. Despite this, we will use circuits constructed with only the Pauli and Clifford gates throughout this thesis.

2.4 Quantum Noise

Having constructed a model of universal quantum computation implemented using a small set of basis gates, we are now drawn to consider the problems that arise from the implementation of

this model. These errors may be recoverable transformations or may move the system outside of the computational space. For instance if the data is stored on the ground or excited state of an ion or on the polarisation state of a photon, the electron or photon might escape. In this thesis we will constrain ourselves errors that do not directly include these 'leakage' errors.

For a given pure state, the action of almost any non-identity operation will result in an error with non-zero probability when the state is measured. The noise itself can take the form of any quantum operation. In a closed system, this noise operation will be unitary.

The Choi-Kraus theorem [52] provides an operator sum decomposition of these error channels, that is for an error channel $\mathcal{E}$

$$\mathcal{E}(\rho) = \sum_i K_i \rho K_i^\dagger, \tag{2.11}$$

where $\sum_i K_i^\dagger K_i \leq 1$. These operators K_i are termed the *Kraus operators*. For the purposes of this thesis, we will only be considering error channels that are diagonal in the Pauli basis. A general Pauli channel can always be expressed where each K_i is proprotional to an element of $\mathcal{P}^n$, and where the constant or proportionality is the square root of the probability p_i. We will also not be considering the possibility of leakage, hence $\sum_i K_i^\dagger K_i = 1$.

One special case of this channel is the quantum analogue to a classical error operation, the bit flip channel. This channel applies a Pauli X operation to the qubit with some probability p, and leaves the state unaffected with probability $1-p$. This channel is given by

$$\rho \to \eta(\rho) = (1-p)\rho + p\sigma_X \rho \sigma_X^\dagger. \tag{2.12}$$

The operation elements are then $\sqrt{1-p}\sigma_I$ and $\sqrt{p}\sigma_X$. This channel flips acts to flip the state of a qubit between $|0\rangle$ and $|1\rangle$, analogous to the classical bit flip channel. Similarly there exists a dephasing channel that applies a Pauli Z operation and has operation elements $\sqrt{1-p}\sigma_I$ and $\sqrt{p}\sigma_Z$. We can also consider the depolarising channel

$$\rho \to \eta(\rho) = (1-p)\rho + \frac{p}{3}\sigma_X \rho \sigma_X^\dagger + \frac{p}{3}\sigma_Y \rho \sigma_Y^\dagger + \frac{p}{3}\sigma_Z \rho \sigma_Z^\dagger. \tag{2.13}$$

This channel applies one of the non-identity Pauli operations with probability p, and each of these Pauli operations occur with equal probability.

A general Pauli error channel over n-qubits can be represented by the sum of the operation elements for all convex combinations of Pauli operations on each qubit given by

$$\rho \to \eta(\rho) = \sum_{\sigma_i \in \sigma^{\otimes n}} p_i \sigma_i \rho \sigma_i^\dagger. \tag{2.14}$$

For an ordering of the operation elements, this channel is described by the vector of the probabilities associated with each of the operation elements $\vec{p}$. We will term this vector an error model. As this vector grows exponentially with the number of qubits in the system, it is often easier characterise the system in terms of an error model on a smaller number of qubits and then attempt to map it to the larger system. Typically this error model exhibits independent and identically distributed noise, as a single qubit channel is applied to each qubit in the state. This approach precludes the inclusion of correlated multi-qubit errors, which are seen in many quantum devices [53].

Attempts to use multi-qubit channels (including correlated noise) to approximate the overall error channel encounter generally fail to accurately estimate the noise [59, 38, 63].

2.5 Classical Error Correction

In classical coding theory, we protect against errors with a map between a binary vector of raw information and a set of encoded binary vectors termed 'codewords'. Errors act as bit flips on these encoded states and can be corrected by mapping the binary vector with errors back to the correct codeword. Linear codes describe a class of codes where any linear combination of codewords is also a codeword.

For these linear codes, the encoding operation can be described by an n by k generator matrix G acting on a binary vector of length k, to give an encoded state of length n. Along with G we can define the parity check matrix P that acts to determine if a given vector is a codeword. P takes the form of an $n-k$ by n matrix where $PG = 0$. For a state s that is a codeword for some unencoded state v, it then holds that

$$Ps = PGv = 0v = 0. \tag{2.15}$$

Hence P annihilates codewords. When we apply some error e, we instead see that

$$P(s \oplus e) = Ps \oplus Pe = 0 \oplus Pe = Pe. \tag{2.16}$$

The action of P on a vector then depends only on the error e. This vector Pe is termed the syndrome. As P is a matrix of size $(n-k) \times n$, there are 2^{n-k} possible syndromes, including the null syndrome 0.

The Hamming distance between two binary vectors is defined as the number of bit flip operations that must be performed to convert one vector to the other. As errors acting on the encoded states have a non-zero chance of mapping one codeword to another one, the weight of the smallest undetectable error is given by the minimum Hamming weight between any two codewords and is termed the distance of the code d. Each codeword may then corrects errors of weight less than $d/2$ in a 'Hamming sphere' around the particular codeword. The code can be said to correct errors of up to weight t if all errors of weight t or less lie within this Hamming sphere.

These three parameters can be used to describe the code; the number of encoded bits n, the number of unencoded bits k and the minimum hamming distance between any two codewords d.

2.6 Quantum Error Correction

Unlike classical error correction where errors are constrained to bit flips, quantum errors can take the form of any completely positive trace preserving operator. The general approach to protecting against errors is to ensure that there is enough redundancy in the encoded state that the original state can be recovered after some noise is applied. However, qubits cannot be cloned[73], which prevents the use of a quantum analogue to the repetition code, and many other classical error correcting codes.

Instead, we can encode information in a linear superposition of states. These states are termed codewords, and the set of all codewords forms our code-space. Operations on the code can then either remain within the code-space, or map out of the code-space, the latter are termed errors. The purpose of a quantum code is to detect when the encoded state has been mapped out of the code space, and return it. Unlike classical codes, where there exists a finite number of errors, an error on a quantum code can take the form of any completely positive trace non-increasing operation, of which there are an infinite number. Despite this difficulty, if a quantum

code corrects errors A and B, it also corrects linear combinations of A and B [34]. If a quantum error correcting code that can correct errors in the linear span of n qubit Pauli errors, it can then correct all n qubit errors.

One class of quantum error correcting codes are stabiliser codes. Here the code space is the $+1$ eigenspace of a set of Pauli operators termed the stabilisers (S)[11]. The code space is then invariant under action of the stabilisers. The code space itself is comprised of a series of codewords, forming the logical space of our code. Maps between these codewords that commute with the stabilisers are termed logical operators and are denoted with a bar $\bar{X}$. These stabiliser codes can be described as an encoding of n physical qubits into k logical qubits.

Classically the Hamming weight is the number of 1 bits in a bit string, here it is defined as the number of non-identity Paulis in a Pauli string. The minimum Hamming weighted non-identity error (number of non-identity Pauli elements in the error) that commutes with all the elements of the stabilisers is defined as the distance of the code. The particular Pauli error string (or set of error strings) is the smallest undetectable error that can act on the code. The Hamming weight of this error then determines the distance of the code d. Each stabiliser code can then be described in terms of $[n, k, d]$; however this is not a unique description, as there may exist multiple codes that share these values.

This stabiliser matrix acts in a similar manner to the classical parity check matrix. Errors taking the form of Pauli operations will anti-commute with elements of these stabilisers and when measured will result in one of three outcomes: either the error will anti-commute with some members of the stabiliser group and be detected; commute with all of the the stabilisers as a member of the stabiliser group and have no effect; or commute with all of the stabilisers and act as a logical error, changing the encoded logical state.

To perform error correction for a stabiliser code, we are only required to measure the eigenvalue of each generator of the stabiliser [66]. This measurement operation projects the linear combination of operations described by the error channel onto an element in the Pauli basis. The measurement results determine which stabilisers the error anti-commutes with and, as all Pauli operations are unitary, they may be inverted and hence corrected. The correction operation is then to apply the Pauli error operation again (or any operation that is equivalent to it modulo the stabiliser group).

The encoding isometry $V : \mathbb{C}^{2^k} \to \mathbb{C}^{2^n}$ acts via $\mathcal{E}[\rho] = V_e \rho V_e^\dagger$. It takes an initial physical space and maps it to an encoded logical space with basis $|\bar{0}\rangle$ and $|\bar{1}\rangle$.

We can similarly define a decoding operation as a projector $W : \mathbb{C}^{2^n} \to \mathbb{C}^{2^k}$ that takes the logical space back to the physical one. As the net action of the encoding operation and the decoding operation on a state should be identity operation, and V is Hermetian, the decoding operation acts via $\mathcal{D}[\rho] = V_e^\dagger \rho V_e$.

After encoding our qubits, they evolve under the action of some noise $\mathcal{N}$ described as a set of discrete Kraus operators

$$\mathcal{N}[\rho] = \sum_i p_i N_i \rho N_i^\dagger. \tag{2.17}$$

We will consider the special case of expanding this general noise map in the Pauli basis, and restricting our noise to positive linear combinations of Pauli operations. These noise operators are then described by $N_j = \sigma_{j_1} \otimes \sigma_{j_2} \cdots \otimes \sigma_{j_n}$ and each operation is applied with probability p_i.

Binary symplectic form provides a short-hand representation of these Pauli operations, by reducing each matrix to a binary check matrix of length $2n$ where the elements in the range 0 to n represent Pauli X operations, while Z operations are found in the range n to $2n$. The index

of the element within each of these ranges indicates which qubit the Pauli operator is acting on. The representation of the Pauli matrices in this binary symplectic form is given by

$$\begin{aligned} I &= \left(0 \quad | \quad 0\right), \quad X = \left(1 \quad | \quad 0\right), \\ Y &= \left(1 \quad | \quad 1\right), \quad Z = \left(0 \quad | \quad 1\right). \end{aligned} \tag{2.18}$$

In this notation, matrix multiplication is reduced to an XOR operation, $\oplus$ while the tensor product becomes a concatenation operation on each of the blocks. For example $X \otimes Z = \left(1 \quad 0 \quad | \quad 0 \quad 1\right)$. Throughout the rest of this thesis we will continue to use the binary symplectic form of each of these objects. It is common to omit the tensor product symbols as a shorthand when writing out these Pauli strings, hence $XZ := X \otimes Z$.

By measuring the stabilisers, the overall noise is discretised and yields a set of syndromes that characterise the commutation relations of the Pauli errors with the stabilisers. These commutation relations take the form of a classical bit string termed the syndrome. For an error E acting on an encoded state with stabiliser group S, the associated syndrome can be represented in binary symplectic form by

$$\vec{s} = EJS. \tag{2.19}$$

Where J is a matrix of the form

$$\begin{pmatrix} 0 & I \\ I & 0 \end{pmatrix} \tag{2.20}$$

.

This symplectic matrix J acts to "twist" the X and Z blocks of the binary symplectic form such that the product of a twisted matrix and a non-twisted one gives the commutation relations between the two.

Along with the stabilisers S, we can also define the destabilisers D. This is a group of commuting Pauli operators where each element of the group pairwise anti-commutes with a single element of the stabiliser group. A method for how to find the destabilisers are presented in chapter 5. Along with the syndrome information $\vec{s}$ the destabilisers can be used to map a state back to the code space. As each syndrome bit represents a stabiliser with which the errored state anti-commutes, by applying the associated destabiliser the state will then commute with that particular stabiliser. This operation acts to map errored states back to the codespace.

By applying the destabiliser for each flipped syndrome bit the we can construct our recovery operation to map the state back to the code-space as

$$R_s = \vec{s}D. \tag{2.21}$$

While this initial recovery operation R_s returns our state to the code space, it can only do so up to some overall logical operation; while the state commutes with all the stabilisers and is in the code space, it may be in the wrong logical domain within the code space compared to its initial position. Analogous to the syndrome, we can determine which logical operators the recovered state anti-commutes with, and determine a logical correction,

$$\vec{l} = (E \oplus R_s)JL. \tag{2.22}$$

Here l is a binary string that indicates which logical operators our state anti-commutes with, and hence what logical operators have been applied by E and R_s. With the convention that the

index in the arithmetic is given modulo $2k$, where k is the number of logical encoded qubits. We can pair off the anti-commuting elements of the logical operators and order them such that JL returns the anti-commuting partner. In a similar manner to the stabilisers and destabilisers, we can then construct a recovery operator for that particular logical error by determining the class of logical error that has been induced.

In the same manner that applying the destabilisers associated with the flipped syndrome bits recovers a state to the codespace, applying the anti-commuting logical operators corresponding to our 'logical' syndrome bits l, will recover any overall logical operation associated with applying $E \oplus R_s$. For example, if the logical operators are ordered $\left[\bar{X}, \bar{Z}\right]$, then the 'logical syndrome' $[0, 1]$ indicates anti-commutation with $\bar{Z}$, which can be corrected by applying $\bar{X}$. The logical recovery operator R_l is then given by

$$R_l = \vec{l}JL. \tag{2.23}$$

As the logical operators commute with the stabilisers, applying these will not map the state out of the code-space again.

Our overall recovery operation for this particular Pauli error can then be given by $R_s \oplus R_l$. This recovery operator will ensure that the resulting state both commutes with all the stabilisers and is hence within the code space, and did not perform an overall logical operation. This can be expressed by

$$(E \oplus R_s \oplus R_l)JS = \vec{s} = \vec{0} \tag{2.24}$$

and

$$(E \oplus R_s \oplus R_l)JL = \vec{l} = \vec{0}. \tag{2.25}$$

It should be noted that all errors that share a syndrome are mapped back to the code-space by applying the same set of destabilisers. However they may not share the same 'logical syndrome'. Conversely, as the space of all syndromes is smaller than the space of all Pauli errors, multiple errors will share the same syndrome. As a result of this, given just the syndrome information we cannot know the exact error that has occurred on the state, only which set of possible Pauli errors. As these errors may not share the same logical class after the application of the destabilisers, we must make a decision between the different choices of R_l for the set of all E_k that share a syndrome. The mechanism that makes this decision is termed a 'decoder'.

2.7 Decoders

A decoder is a map from the syndrome information to a set of recovery operations that attempts to correct errors acting on an encoded state. Decoding can be considered a two step process: first the state must be returned to the code space, and second any overall logical error that has occurred as a result of the first step must be corrected.

By applying the relevant destabiliser operators associated with the syndromes that the error anti-commutes with, an error may be returned to the code space modulo some overall logical operation. The syndrome information can then be used to attempt to determine what physical error occurred, and as a result apply a logical operation to correct for any logical error that occurred when applying the destabilisers. This map from physical errors to syndrome classes and logical classes is predicated on the choice of code.

As the set of all possible syndromes is strictly smaller than the space of all possible Pauli errors, and given that not all errors that share a syndrome class share a logical class, there exist errors for which the wrong logical class will be applied by a decoder. There are thus many

different decoding strategies, and the performance of each depends on both the distribution of errors and the choice of code [21].

Our goal is then to construct a set of recovery operators $\mathcal{R}$ associated with syndromes s such that upon the measurement of a syndrome s_i operator $\mathcal{R}_i$ will return it to the code space and the correct logical class with the highest probability.

For all errors E_s that share a syndrome, we can construct a probability distribution across the set of potential logical recovery operations. From each syndrome class, the maximum likelihood decoder selects the most probable logical error from this distribution as a recovery operation. The final recovery operation associated with a particular syndrome is then given as $\mathcal{R}_i = \mathrm{ML}(R_l) \oplus R_{s_i}$ as R_{s_i} is fixed for each syndrome.

Given a set of errors with the same syndrome s_i, the maximum likelihood logical class can be selected by

$$\vec{l}^{\star} = \mathrm{argmax}_l \left(\sum_{E_{s_i}} p(E_{s_i}) R_l(E_{s_i}) \right), \tag{2.26}$$

where we treat the argument as a binary vector of the logical operations that constitute R_l. Using equation 2.23, $R_{l^\star}$ becomes the logical recovery operation for this particular syndrome. The overall recovery operation associated with this class would then be given as

$$\mathcal{R}_i = R_{l^\star} \oplus R_{s_i}. \tag{2.27}$$

In order to determine this probability distribution over the logical classes, we require an accurate model of the probabilities with which each of these Pauli strings E_k occur, and the ability to numerically iterate over the set of all potential Pauli errors. In general this set of errors grows with the number of qubits as 4^n, and may only be estimated for large systems.

2.8 Fault Tolerance

Within a circuit, errors may spread such that an initial error on a single qubit can effect multiple qubits. For instance, a Pauli Z error on the target qubit of a $\wedge_1(X)$ (a CNOT gate) will spread to the control qubit, while a Pauli X error on the control qubit will spread to the target. The spread of low weight errors can then lead to higher weight errors ultimately exceeding the distance of the code, causing an overall logical error [48]. Fault tolerant quantum computation is an approach to the design of quantum circuits that attempts to limit and correct the spread of these errors[35].

Once the logical error rate under quantum error correction exceeds the physical error rate, it becomes possible to expand the size of the system and exponentially decrease the failure rate of the encoded information. With these lower error rates it is then feasible to perform arbitrarily long quantum algorithms on noisy devices[5, 4, 17, 47].

Within this thesis we adopt the following definition of fault tolerance:

An error correction protocol is t fault tolerant if:

1. for a codeword with Hamming weight w and an error with overall weight s, $w + s \leq t$, and
2. s faults occur during the protocol where $s \leq t$, and the output state differs from a code word by at most weight s[12].

The first condition ensures that a correctable error does not spread to become an error with a weight greater than the distance of the code, becoming uncorrectable. The second condition

ensures that repeated rounds of an operation do not propagate errors uncontrollably. The application of a measurement circuit for a single stabiliser is fault tolerant if the weight of any Pauli errors at any location on the circuit modulo the stabiliser group does not result in an overall Pauli error with a greater weight on the code block.

A bad location is defined as a location on the circuit in which a single fault error will result in a Pauli error on the code block with weight greater than two. We can consider an example of a bad site in a syndrome measurement circuit: a Pauli Z error acting on an ancilla qubit will propagate back into the code block with each subsequent CNOT operation where the ancilla qubit is the target, this can be seen in figure 2.6. One method of implementing fault tolerant measurement circuits requires that we must be able detect and correct errors at all bad locations on the ancilla qubits.

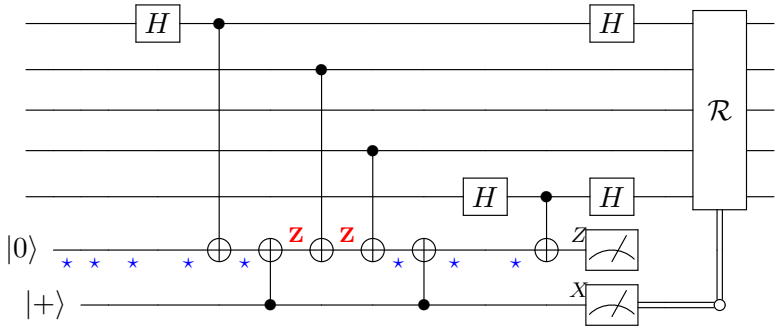

Figure 2.6: Flagged fault tolerant circuit for measuring the stabiliser $ZXIXZ$. The first five qubits comprise our register block, the qubit prepared in the $|0\rangle$ state is an ancilla while the qubit in the $|+\rangle$ state exists to flag the existence of a fault in a location that would result in a higher weight error. The bad locations are marked with red Z's. Any error in a location that would spread to the code block with weight greater than two is also spread to the flag qubits below. These errors can then be diagnosed and corrected.

In the above circuit, Pauli Z errors on the ancilla qubit would spread to the register block when a CNOT operation is applied. Under the criteria for fault tolerance, we must suppress this spread of errors such that a single error at a single location leads to at most a single error in the register block. Errors at locations denoted by blue stars will spread to the register block, but will produce either weight one errors in the register block (as can be seen most easily by the right most location giving a single X error in the register block). Errors at locations with red Z's will instead produce a weight two or three error. When considering these errors modulo the stabiliser, these are both weight two errors which cannot be corrected by the five qubit code. However, an error in this particular location can be detected by the flag qubit, and a recovery operation may then be performed to reduce the weight of the error to one. As the errors share two common elements, the $IZIZX$ or the $IIIZX$ operation may be performed, completely correcting one of the errors and leaving the other at weight one.

When considering a fault tolerant operation over multiple encoded blocks on a quantum device, a single error should spread to be at most one error per block, and each block should be able to correct that error. This property can be fulfilled by operations that include no gates that act on multiple qubits within the same block, and that no gate between two blocks act on the same qubit (this prevents a single error on that qubit spreading to two qubits on the other block). Such operations are termed transversal operations. While a transversal operation

is fault-tolerant, though not all fault tolerant operations are transversal.

Chapter 3

Noise and Circuit Simulation

In the previous chapter we outlined stabiliser based quantum error correction and fault tolerance, which are required for the implementation of a practical quantum memory. This chapter addresses the implementation of error correction and fault tolerance using Clifford circuits.

Using these gates, stabiliser circuits can be constructed from just the generators of the Clifford group[1, 36]. Hence we can decompose our encoding, decoding, measurement and recovery operations into applications of these gates. In addition, the Cliffords, as the normaliser of the Pauli group act as a map between probability distributions of Pauli operators. Our Clifford circuit simulation can then be used to propagate distributions of Pauli errors represented using Kraus operators through Clifford gates, and produce a final set of Kraus operators that characterise the total error channel of the circuit for a particular error model.

As previously shown, with a completely characterised error model, we can construct maximum likelihood decoders. We will also demonstrate later in this chapter how to search over the space of stabiliser codes to find codes that better tolerate a particular noise model. Modern quantum devices often exhibit errors associated with each operation on the device, and so we need to 'unravel' the overall error model from this combination of operations and local noise [39]. This unravelling procedure can be see in figure 3.1.

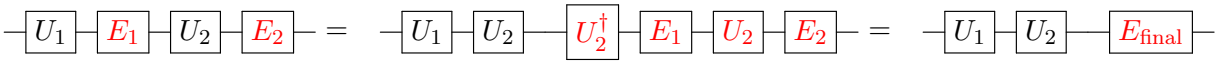

Figure 3.1: The transformation to extract an error model from a set of errors associated with unitary operations acting on the state. For large systems, constructing the matrices and calculating the commutation relations required to find the explicit form of E_{final} is prohibitively computationally intensive. Instead we attempt to simulate the action of the unitaries on the distribution of errors on the state to construct the set of Kraus operators for E_{final}. This circuit reads left to right.

However, the size of these matrices scales exponentially with the number of qubits and it quickly becomes computationally infeasible to use these objects directly. Instead we note the sparseness of these matrices. We can constrain our operations to the Cliffords generators and the Paulis, which only ever act on one or two qubits. Similarly our noise can be decomposed into the set of Kraus operators and applied linearly to elements of our super-operator, instead

of constructing the entire channel. With the exception of high weighted errors, we then consider these sparse, block diagonal matrices and perform the associated gate in a more computationally efficient manner.

3.1 Circuit Simulation

The effect of a quantum gate or noise can be described by its action under conjugation on the density matrix representing the state of the system. These take the form of two $2^n \times 2^n$ matricies where n is the number of qubits. For $n > 10$ storing these objects quickly exceeds the size of the cache on most current computers leading to significant slowdowns, while for $n > 15$ it may exceed the total amount of memory on the device. The best current matrix multiplication algorithms over complex numbers achieve a bound of approximately $O(2^{2.4n})$ per matrix multiplication. In this section we will present a more memory efficient and computationally efficient approach for applying gates and noise to the circuit for Clifford and Pauli operators.

Starting with our Kraus operators in the Pauli basis, for each element we have both a Pauli string and an associated probability. An operation on this Kraus operator can then be described by its action on the Pauli string, and its action on the associated probability. For instance, a Pauli X will preserve the associated probability while modifying the Pauli string. Each of these update procedures will be comprised of an operation on the binary symplectic representation of the Kraus operator and a completely positive map that dictates the distribution of the probabilities. We will see in Chapter 5 that we can completely dispose of storing the Kraus operation elements themselves by a fixed ordering of the probabilities.

To simulate the action of the Pauli and Clifford gates we implement the following update operations on binary symplectic strings.[1]. X and Z denote the respective blocks of the binary symplectic vector with subscripts for the blocks in column major order. σ indicates a Pauli operator in binary symplectic form with superscripts indicating the X or Z block. As these operators are strings, a single subscript indicates the position within the string. When acting on a set of Kraus operators, i then denotes the operation element, while when acting on the tableau formalism i is the row.

1. Pauli operations: As matrix multiplication is equivalent to the XOR operation in this formalism, the binary symplectic Pauli operation σ_b can be performed on some qubit a by $X'_{i,a} = X_{i,a} \oplus \sigma_b^X$ and $Z'_{i,a} = Z_{i,a} \oplus \sigma_b^Z$ for all i.

2. CNOT: For control qubit a and target qubit b, $X'_{i,b} = X_{i,a} \oplus X_{i,b}$ and $Z'_{i,a} = Z_{i,a} \oplus Z_{i,b}$ for all i.

3. Hadamard: We map any X error to a Z error and any Z error to an X error. This can be represented by the following operation acting on qubit a, $X'_{i,b} = Z_{i,b}$ and $Z'_{i,b} = X_{i,b}$ for all i.

4. Phase: When applying a phase gate to some qubit a we set $Z'_{i,a} = Z_{i,a} \oplus X_{i,a}$ for all i

We can concern ourselves with only 'tracked' Kraus operators where $p_i > 0$ as any update on a Kraus operator where $p_i = 0$ will have no net effect on the overall distribution. From an initial distribution of $p_I = 1$, each application of a gate potentially grows the number of tracked Kraus operators up to the full range of 4^n Pauli strings.

An IID Pauli error on a single qubit can then be itself represented as a Kraus operator in the Pauli basis with operating elements and associated probabilities. This IID error can then

act on our current distribution by applying each of the binary symplectic Pauli operators from this Kraus operator and then tracking the probability of each of the resulting Pauli strings. An IID error will grow the number of tracked Kraus operators.

We must also consider state preparation and measurement. As we are only tracking the action of Pauli operations, we cannot explicitly distinguish between states. Instead we restrict this to instances where the state preparation and measurement are in the same basis and are only subject to Pauli and CNOT operations.

1. State preparation on some qubit a by setting $X'_{i,a} = 0$ and $Z'_{i,a} = 0$, then tracking the action of the Pauli operations on this state.

2. This same qubit, a, can be measured in the Z basis by setting $Z'_{i,a} = 0$ and returning $X_{i,a}$, similarly a measurement in the X basis can be performed by setting $X'_{i,a} = 0$ and returning $Z_{i,a}$. We must then determine what the effect of this Pauli error is on the expected outcome.

As a simple example; we prepare an ancilla qubit in the $|0\rangle$ state, and check when measuring whether any overall Pauli X noise has been applied to that particular qubit and flipped it. Obviously this approach will not work with non-parity operations between tracked and untracked states; for example the state of the code block is not known while the state of the ancilla qubits can be determined.

Given that we are now mostly concerned with tracking the probabilities associated with each of these Kraus operators, we can treat each binary symplectic state as a binary string. The representation of this string under $\mathcal{Z}^+$ can then be used as the index of a vector p of the probabilities of each of the Kraus operators from equation 2.17.

Each of the above gate operations can now be considered a linear operation on p. This gate operation must map from an initial Pauli string to some set of Pauli strings, and must be a completely positive map that preserves the total probability associated with the initial Pauli string.

If we consider the action of G on a single element of p such that $[i'_1 \dots i'_k] = G(i)$, the requirement that G be a completely positive Unitary operation we can infer that

$$p_i = \sum_{j \in G(i)} p'_j. \tag{3.1}$$

As this holds for each element of p the probability distribution of each element is preserved and $\sum p = \sum p'$.

The update operation for some gate G and associated binary symplectic index i is then shown by figure 3.2.

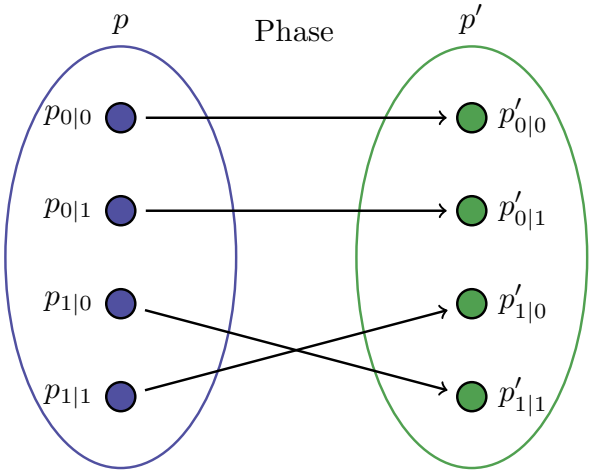

Figure 3.2: An update of the probabilities associated with the Kraus operators p, by the Clifford phase gate. Clifford and Pauli gates act as injective maps that permute the input state, while Pauli noise operations with non-zero elements in their probability distribution are block-wise complete bipartite graphs.

Error simulation on each of the gates is then performed by applying an error update following each of the gate updates as shown in figure 3.4. As previously established within this formalism, these errors must take the form of a Kraus operator in the Pauli basis.

$$p_i = \sum_{j \in G(i)} p'_j \tag{3.2}$$

3.2 Code Searching

Stabiliser codes are discrete entities and may be sampled over [63]. This sampling allows us to 'search' across codes and compare their performance under different error models.

We will first consider the standard form for a stabiliser code as a binary symplectic matrix divided into a number of smaller 'blocks'[52]. We can begin by generating a stabiliser matrix of the form

$$\left[\begin{array}{ccc|ccc} I & A_1 & A_2 & B & 0 & C \\ 0 & 0 & 0 & D & I & E \end{array}\right]. \tag{3.3}$$

Where all non-identity and non-zero blocks (A, B, C, D and E) are initially populated with randomly generated binary values. It can be seen that even in the case that all of these elements are zeroed, the identity elements adjacent to a block of zeros ensure that there are no degenerate pairs of stabilisers. This also ensures the commutation of the stabilisers in this particular case, but is not true for the general random population. We will next show a method of fixing choices of these sub-blocks to ensure commutation.

We start by filling (A_1 to E) randomly. Now that we have no degenerate pairs of stabilisers we need to ensure that all stabilisers commute. We will first consider the stabilisers S_1 to S_r, or the top row of sub-blocks. For stabilisers $S_i, S_j, i \neq j$ elements within A_2 and C may anti-commute, and elements within B and I may anti-commute. We can then leverage the commutation relation

between B and I to ensure that the stabilisers commute; if S_i, S_j anti-commute then flipping $B_{i,j}$ will cause an additional element-wise anti-commutation between B and I, ensuring an overall commutation between the two stabilisers.

As the stabilisers from S_{r+1} to S_{n-k} (the bottom row of sub blocks) are comprised of only Z elements, they are already guaranteed to commute amongst themselves. Our only consideration is now between elements within the two groupings of stabilisers. For some $S_i, S_j, i \in [r+1, n-k], j \in [1, r]$ we can use the same argument between B and I to flip $D_{i,j}$ if S_i and S_j do not commute.

At this point all stabilisers in our random code commute and we turn our attention to the logical operators. Using the same blocks as previously, we can construct logical operators as

$$\begin{array}{c} \begin{array}{ccc} \bar{X} & & \bar{Z} \end{array} \\ \left[\begin{array}{c|c} 0 & 0 \\ E & 0 \\ I & 0 \\ - & + \quad - \\ C & A_2 \\ 0 & 0 \\ 0 & I \end{array}\right] \end{array}. \tag{3.4}$$

Once again we note that the inclusion of identity elements ensures that the logical operators are non-degenerate independent of the choice of C and A_2.

Here we must consider the commutation relations both within the logical operations and between the logical operations and the stabilisers. In particular we require pairwise anti-commutation between the $\bar{X}$ and $\bar{Z}$ blocks, but commutation between the logicals and the stabilisers.

As the only anti-commuting elements between $\bar{X}$ and $\bar{Z}$ are between two identity blocks, we have pair-wise anti-commutation between $\bar{X}_i$ and $\bar{Z}_i$, while all other logical operators commute with the associated 0 block.

We now must ensure that the logical operators commute with all of the stabilisers; and for this we can use the C and A_2 blocks.

Considering first the elements of $\bar{Z}$, I anti-commutes with A_2 from equation 3.3 and conversely A_2 anti-commutes with I, ensuring commutation between these blocks. Similarly, by construction we note that with the first r stabilisers $\bar{X}$ have anti-commutation relations between I and C and C and I blocks, once again ensuring an overall commutation, for $S_{[r+1,n-k]}$ we have anti-commutation relations between E and I and I and E.

Hence we now have pair-wise anti-commutation within the logical operators, commutation within the stabilisers and commutation between the logical operators and the stabilisers along with ensuring that all stabilisers and logical operators are non-degenerate. Lastly we note that by placement of the identity blocks and zero blocks, that our operators have been constructed such that all stabilisers and logicals are independent.

3.3 Circuit Noise

While we have so far discussed errors in generality, or in single instances of Paulis, it bears pinning down the errors we will be discussing. An error model is a description of each error that may occur and the probability with which it occurs. For Pauli errors, it be represented as a Kraus operator in the Pauli basis, but with some simplifying assumptions more compact

representations are possible. We will be constraining ourselves to Pauli errors for the remainder of this thesis.

An independent and identically distributed (IID) error, is one that applies errors with equal probability between types of error (in this case non-identity Paulis) with equal probability over qubits. The associated error model can be exactly characterised by the probability of a non-identity Pauli p_e and the number of qubits it acts on n.

$$P(E) = (p_e)^{\text{Hamming(E)}}(1-p_e)^{n-\text{Hamming(E)}} \tag{3.5}$$

While this is a noise model to work with, we can also consider a range of noise models. By picking some physical error rate $p_e = p_x + p_y + p_z$, there is a freedom in the choice of the probability with which each of the non-identity Pauli operations occur. These probabilities can be represented by the vector

$$\vec{P} = [1-p_e, p_x, p_y, p_z]. \tag{3.6}$$

In the case that $p_x = p_y = p_z = \frac{p_e}{3}$, we have the depolarising channel (our IID channel from above), while if $p_x = p_y = 0$ and $p_z = p_e$ we have the dephasing channel. Paramterising these two channels, we define a bias $\eta > 0$ that dictates the rate of Pauli Z errors relative to X and Y errors such that $p_x = p_y = \frac{p_z}{\eta}$. This error model can be described as

$$\vec{P_\eta} = [1-p_e, \frac{p_e}{2+\eta}, \frac{p_e}{2+\eta}, \frac{\eta p_e}{2+\eta}]. \tag{3.7}$$

For $\eta = 1$ the paramaterisation takes the form of the depolarising channel, while as $\eta \to$ inf we see it approaches the dephasing channel. Previous experiments have demonstrated error channels that approximately lie within this range, with values of η ranging up to nearly 100[53].

Optimistically, the only source of errors acting on a given quantum system would be some well characterised and understood noise with a uniform identical distribution over the qubits in the code and no gate, preparation or measurement errors. With these simplifying assumptions we have discarded any comparison that can be drawn between different codes and their encoding, decoding, syndrome measurement, recovery or transversal gate circuits.

Even with this simple approximation of the errors that can act on a circuit, tailoring a maximum likelihood decoder to the error model can result in significant improvements in the overall error rate, especially if a code is found such that errors that occur with high probability share both a syndrome class, and a logical class [63].

The action of this optimistic description of the action of errors can be described by figure 3.3.

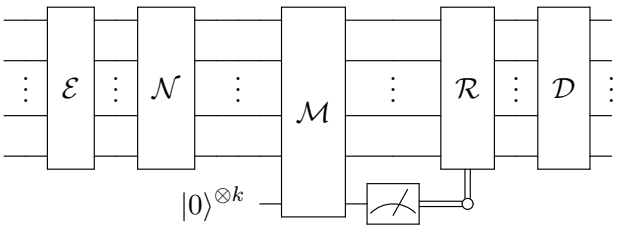

Figure 3.3: An ideal error correction scheme. A noiseless encoding circuit $\mathcal{E}$ is applied before some well characterised noise source $\mathcal{N}$ acts on the encoded state. The resulting state is measured using a noiseless measurement circuit $\mathcal{M}$ and is then recovered by passing the classical syndrome information to a decoder and informing the recovery circuit $\mathcal{R}$. The encoding operation can then be reverse by $\mathcal{D}$

An extension of the above ideal model is to consider the decomposition of the encoding and decoding circuits into a set of noisy Clifford operations. The noise on the Clifford gates within these circuits can be represented by the action of a noisy identity gate on each qubit that does not participate in the Clifford gate, and by the action of another noisy identity gate over the participating qubits after the Clifford has been applied, this can be seen in figure 3.4.

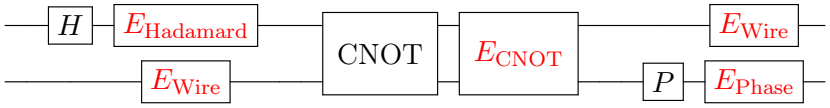

Figure 3.4: A noisy Clifford circuit. Error channels are applied after every Clifford gate, Pauli gate, preparation and measurement operation. These errors are not necessarily the same operation; in experimental setups error rates on two qubit gates may be orders of magnitude larger than those on single qubit gates.

This model relaxes the assumption that the errors are identically distributed across the qubits; each gate can be associated with a different error channel such that the errors applied on each qubit now vary. Multi qubit gates may now also spread and correlate errors throughout the circuit. The error channel introduced by this model has a rate proportional to the number of gates in the circuit. Additionally, the error rate associated with different Clifford generators may vary, such that circuits that preference particular Clifford gates may have a lower overall physical error rate.

Throughout this thesis, we will focus our attention on the interleaved errors, where each error can be described by an error model over a subset of the qubits in the circuit; and different error models may be applied for different gates and wire errors.

Chapter 4

Circuit Construction

Having determined how to construct the error channel from a noisy circuit, we now consider what circuits to apply this technique to. In the context of quantum memories we consider the encoding circuit, measurement circuit and the decoding circuit. As these Clifford gates map probability distributions over the Pauli operators to probability distributions over the Pauli operators, we can characterise the circuit in terms of the distribution of Pauli errors for a given error model.

By searching over a range of different circuits using a fixed error model, we can attempt to minimise the overall error rate associated with that particular operation. In particular we can search across a range of random codes generated as shown in section 3.2 for a given error model in an attempt to find a code with an associated circuit that produces a lower overall error rate after correction. These error models can be weighted such that certain elements of the Clifford group have higher weights than others, or such that different individual physical qubits exhibit different error rates. As the Kraus operators propagate through the circuit, multi-qubit gates will spread and correlate the errors.

For an arbitrary random stabiliser code we must now find a method of generating the Clifford circuits associated with encoding, measurement and decoding.

4.1 Encoding Circuits

Our goal is to construct a set of Clifford gates that will map our stabilisers to the identity and act as a decoding circuit. Performed in reverse we have constructed an encoding circuit. This section will make extensive reference to the notation and operations described in section 3.1, in particular the use of Clifford operations on binary symplectic strings.

We will make repeated use of two processes throughout this algorithm; binary symplectic Gaussian elimination, and the identity $PHASE(I) + B = MM^T$ where M is lower triangular, and M^T is upper triangular. The $PHASE(I)$ operation gives us a free choice of diagonal elements of the identity matrix to contribute to B.

Gaussian elimination is possible as the binary symplectic CNOT acts as an XOR operation between two columns, hence after establishing an 1 on the diagonal entry of the sub-block, it becomes possible to use CNOT operations to clear the rest of the row. So long as a code is non-degenerate, if there is not a 1 element in the diagonal entry, one can be established by using a CNOT operation within the same block, or a Hadamard operation followed by a CNOT if there only exist 1 elements in the adjacent block. By clearing each row sequentially, later rows may operate left of the diagonal without eliminating prior diagonal elements. Our second identity uses phase gates to map elements from the diagonal of the X block to the Z block to construct B such that the identity holds. Having already set the X block to the identity with our Guassian

elimination, we are always free to perform these operations in series for non-degenerate codes.

For each stabiliser in our random code there exists an associated destabiliser D. These stabilisers and destabilisers pairwise anti-commute. We will start by constructing a tableau matrix of the stabilisers and destabilisers [1].

$$T = \left[\begin{array}{c|c} D^X & D^Z \\ S^X & S^Z \end{array}\right]. \tag{4.1}$$

This can be formed by stacking the rows of the stabilisers and destabilisers in each block. The stabilisers and destabilisers should exhibit the same pairwise ordering of anti-commuting elements as the logical operations, discussed in section 2.6. That is to say that for each row i stabiliser S_i should anti-commute with the destabiliser D_i.

Though the stabiliser sub-matrix S is full rank, the S^X and S^Z sub-matrices may now still be rank deficient. The solution here is to construct an identity submatrix in S^Z and then use Hadamard operations on the last k qubits to map elements from the S_Z block and create an identity sub-matrix on the S^X block. In the case of random codes, this can be achieved by performing Gaussian elimination using CNOT gates.

Now that the S_X matrix has full rank we can perform symplectic Gaussian elimination using CNOT operations. Our Clifford operations acting on binary symplectic vectors have been described in section 3.1. In particular we will use the ability to perform 'in block' XOR operations to create a diagonal element in each row, then use that element to eliminate all other elements in the row. So long as the stabilisers are not degenerate, then it should be impossible to completely eliminate a future row, for a discussion of why the randomly chosen stabilisers are fixed such that they are not degenerate, see section 3.2 . The Clifford based Gausian elimination algorithm then follows:

- For each row i check if $S^X_{i,i}$ is 1
- If it is not then search for the first nonzero element $j > i$ (if $j < i$ then you will undo your operations on previous rows) and CNOT(j, i)
- Next, for each element in the row $j \neq i$ if j is 1 then CNOT(i, j), as you will have already set all the previous elements in the column to zero using this method you are assured to preserve values in all previous rows.
- Iterate to the next row

This reduces the S_X sub-matrix to

$$T = \left[\begin{array}{c|c} D^X & D^Z \\ I & S^Z \end{array}\right]. \tag{4.2}$$

Our next step is to eliminate S^Z completely. Once this is done we can flip our identity matrix in S^X to the Z block using Hadamard gates. The approach here is to fix $S^Z = S^X$, to achieve this we need to find a matrix M such that $S^Z + \text{PHASE}(S^X) = MM^T$.

For simplicity we will assume that M is lower triangular (hence M^T is upper triangular) such that $S^Z_{i,j} = \sum_{k<j} M_{i,k}M_{j,k} + M_{i,j}$ for all $k > j$ the result will always be zero. We can iterate through each i, j solving for M, and applying a phase gate when required to fix the diagonal element of S^Z. This can be achieved by applying phase gates to set the diagonal elements then using CNOT gates to eliminate the upper right elements of the S_Z block.

As we've now set $S^Z = MM^T$ we can use CNOTs such that $S^X = M = S^Z$, and the tableau takes the form

$$T = \left[\begin{array}{c|c} D^X & D^Z \\ M & M \end{array}\right]. \tag{4.3}$$

In this case as we're generating M from the identity using CNOT operations, we are effectively applying $M^{T^{-1}}$ to the S^Z sub-matrix. Given $S^Z = MM^T$ we are left with $S^X = S^Z$. Take note of any of these CNOT operations being performed, if there is an existing CNOT operation as the last operation performed on the same pair of qubits (with the same control and target) then you can eliminate both in your list of gates.

From here, simply applying phases to each of the qubits results in $S^Z = S^Z \oplus S^X$, given $S^Z = S^X$ then we have eliminated the entire S^Z sub-matrix. As all the previous operations have also preserved the pairwise anti-commutation relations between the D and S sub matrices we note that as S^Z is now 0, any anti-commutation must occur between D^Z and S^X.

After applying phases, the tableau takes the form

$$T = \left[\begin{array}{c|c} D^X & D^Z \\ M & 0 \end{array}\right]. \tag{4.4}$$

As with our first step, we can perform binary symplectic Gaussian elimination again and $S^X = I$. By the same argument as above, as D_i and S_i must anti-commute, and S_i^X is 0, D^Z must anti-commute with S^X. After binary symplectic Gaussian elimination, $S^X = I$, hence D^Z must also be the identity matrix. Our tableau now takes the form

$$T = \left[\begin{array}{c|c} D^X & I \\ I & 0 \end{array}\right]. \tag{4.5}$$

From this position we can now mirror the steps used previously for our stabiliser blocks on our destabiliser blocks. We can skip the first round of Gaussian elimination by flipping the X and Z blocks using Hadamard gates.

$$T = \left[\begin{array}{c|c} I & D^X \\ 0 & I \end{array}\right]. \tag{4.6}$$

Once again solve for $D^Z + \text{PHASE}(D^X) = NN^T$. By the same argument as above, we now set $D^X = M, D^Z = M$. These operations will also act on the S^Z sub-block such that it likely is no longer the identity.

$$T = \left[\begin{array}{c|c} N & N \\ 0 & C \end{array}\right]. \tag{4.7}$$

Again perform phase gates to eliminate D^Z leaving us with just elements in D^X and S^Z;

$$T = \left[\begin{array}{c|c} N & 0 \\ 0 & C \end{array}\right]. \tag{4.8}$$

And finally a third round of symplectic Gaussian elimination on both of these sub-matrices leaving us with just the identity matrix

$$T = \left[\begin{array}{c|c} I & 0 \\ 0 & I \end{array}\right]. \tag{4.9}$$

The list of gates we have applied now form a Clifford circuit from the state stabilised by S to $|0...0\rangle$. Applying these gates in reverse order forms an encoding circuit.

4.2 Measurement Circuits

The construction of a parity check measurement of a stabiliser is relatively straightforward. For each stabiliser we start by preparing an ancilla in the $|0\rangle$ state. Our CNOT operation can propagate a σ_X error from the code block to our ancilla, and by changing the basis of the qubit in the code block before performing the CNOT operation we can propagate Y and Z errors.

$$X = HZH^\dagger = PPPYP^\dagger P^\dagger P^\dagger \tag{4.10}$$

Using these single qubit Clifford gates we can now map our Pauli X errors and apply them to our ancilla qubit using a CNOT gate. After applying the CNOT we then apply the inverse operation to return the qubit to its original state. We repeat this procedure for each qubit that contributes to a particular stabiliser and then measure the ancilla to extract the syndrome bit associated with particular error.

Converting the basis back after each measurement is a somewhat costly in the number of gates, and can be reduced without too much difficulty. Instead of applying the inverse operation, at the end of each stabiliser measurement we track the current basis of each qubit. If the next stabiliser measurement that the qubit participates in is in the same basis we don't need to do anything, otherwise we determine the Clifford operation required to map from the current tracked basis to the measurement basis.

A further reduction is possible by determining the optimal ordering of stabiliser measurements that minimises the number of basis changes required. For example is reasonably evident that the Steane code has a number of obviously sub-optimal stabiliser measurement orderings such as $S_1^X, S_1^Z, S_2^X, S_2^Z, S_3^X, S_3^Z$. Here we require 24 Hadamard gates in basis changes to perform all the stabiliser measurements, while $S_1^X, S_2^X, S_3^X, S_1^Z, S_2^Z, S_3^Z$ requires only 14.

It is further possible to find an optimal ordering of the stabilisers that minimises the number of gates required for basis changes in the measurement circuit, a naïve approach to this problem scales as $O(n!)$ with the number of stabilisers.

4.3 Flagged Fault Tolerance

Having simulated CNOT gates, errors can now spread in a non-fault tolerant way throughout the code qubits [10]. In order to try and suppress the spread of these errors, we now need to attempt to modify our circuits to incorporate fault tolerance. We will apply flagged fault tolerance to our measurement circuits.

In measurement circuits, our particular points of concern are the ancilla qubits. Z errors on ancillas will propagate along all subsequent CNOT operations back into the code block, and in nearly all cases, produce a higher weight error.

To detect these errors, we add additional 'flag' qubits[12, 48], prepared in the $|+\rangle$ basis and perform parity check operations using CNOT gates across each CNOT gate that operates on the code block. A Z error in the ancilla block will now propagate to the flag qubits along with the code block, and 'flip' the flag state such that the eigenvalue of the measured qubit is now -1. The measurement results of these flag qubits can now be treated in much the same manner as

the syndromes and provide information about when an error occurred during the measurement circuit.

When a flag is tripped, syndrome measurement is halted, a flag recovery operation occurs, and the syndrome measurement restarts. So long as the gate error rate is sufficiently low as to preclude an average case of more than one error per syndrome measurement circuit, weight one errors will be contained and suppressed.

We can consider the example in figure 4.1, which presents the measurement of a $ZXIXZ$ stabiliser on a five qubit code. Our measurement sub-circuit then consists of a series of single qubit Clifford operations that perform the relevant change of basis and a CNOT operation to the ancilla, and then the change of basis is reversed. Faults on the ancilla can be considered in terms of Pauli X and Pauli Z errors. X errors flip the state of the ancilla, while Z errors spread along CNOT operations. For a sufficiently low gate error rate, X errors are handled by taking a majority vote over multiple rounds of measuring the same stabiliser.

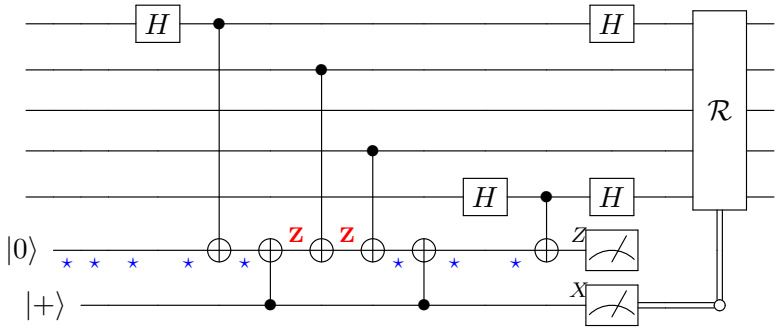

Figure 4.1: Flagged fault tolerant circuit for measuring one of the stabilisers of the 5 qubit code. The bad locations are marked with red Z's. Any error in a location that would spread to the code block with weight greater than two is also spread to the flag qubits below. These errors can then be diagnosed and corrected. The blue stars represent all potential spreadable environmental error locations on the ancilla qubit.

From the fault tolerance conditions, we can permit a single fault to produce an at most weight one error modulo the stabiliser group. A Z error that occurs before the first CNOT operation between the code block and the ancilla qubit will spread a Z operation to each qubit that participates in the stabiliser measurement. When the change of basis is undone at the end of the stabiliser measurement, these Z errors will take the form of the stabiliser across all participating qubits. As a result, we can discard these initial Z errors as having no net effect on the code-space.

A single fault that occurs between last and second last CNOT operations will propagate a single error into the code block and can hence still satisfies the fault tolerance conditions. Likewise these conditions are satisfied by a fault between the first and second CNOT operations will spread a single error modulo the stabiliser element.

For our five qubit example we are only left with the second and third CNOT operations to consider. A single fault within this region on the ancilla qubit would spread a weight two error to the code block and breach the fault tolerance conditions. To detect this error we incorporate a flag qubit prepared in the $|+\rangle$ basis and a pair of CNOT operations to perform a parity check operation over this block.

Should a Z error occur within this region it will spread to both the code block and our flag

qubit, flipping the state of the flag. Measuring the flag qubit then diagnoses this error and we can attempt to correct it. For our five qubit example, the error associated with this flipped flag can be corrected by either a $IZIZX$ or a $IIIZX$ operation. The former completely correcting a weight 3 error (such as may have occurred at the first bad location) and partially corrects a weight two error to a weight 1 error (satisfying the FT condition), while the latter completely corrects the weight two error and reduces the weight three error (weight one modulo the stabiliser) to a weight one error.

We will constrain our stabiliser measurement sub-circuit such that all CNOT operations are performed with the control qubit in the code block and a target qubit as the ancilla. We will also reverse our change of basis after performing the flag recovery, hence all errors spread by errors on the ancilla will take the form of Z errors on the registers. These constraints are simply for our model, and are not required in generality; but will provide some minor advantages in exchange for an increased gate count.

For ease of implementing this method on stabiliser measurements for random codes, we will adopt a more redundant approach. For a stabiliser of weight w we prepare $\log_2(w)$ flag qubits. We will then interleave CNOT operations from the flag block to the ancilla between the CNOT operations from the register block to the ancilla such that each CNOT operation from the register block is uniquely bounded by parity checks to the flag qubits. These parity checks operations then distinctly diagnose any non-SPAM single Z fault on the ancilla. This setup can be seen for the $ZXIXZ$ stabiliser in figure 4.2.

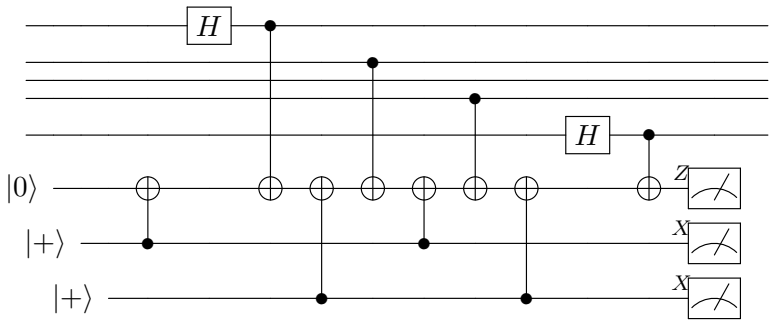

Figure 4.2: Flagged fault tolerant circuit for measuring one of the stabilisers of the 5 qubit code. This more verbose method is applied to the stabilisers of random codes for ease of implementation. This can be verified in appendix A.

Under this setup, all Z errors on the ancilla block simply become Z errors on the register block. This simplifies determining what error occurred as we now only need to track what qubits in the register were measured, not what operations occurred afterwards.

Secondly, the greater number of flag qubits permits us to suppress half the Z errors, rather than merely restricting them to a weight one error on the code block. Within each bounded CNOT operation from the register to the ancilla, an error may occur before the CNOT operation and spread to the code block on the operation, or it may occur after the CNOT operation and only spread on subsequent operations. Each 'flag syndrome' then applies to one of two different errors on the register block that differ by one Pauli Z. Correcting either error completely corrects one of these errors, and suppresses the other to a weight one error.

In particular, one error may have occurred after the flag parity check, but before the CNOT operation to the code block, whereas the other may have occurred after the CNOT operation to

the code block, but before the second flag parity check. As both these errors are bounded within the same CNOT operations to the flag qubits, they will share a the same flag syndrome. There then exists a choice corrections; either by applying Z operations to all 'downstream' register qubits; or applying Z operations to all all 'downstream' and the current register qubit; in which case if it occurred before the current CNOT it is corrected. In the former case, if the error occurred after the CNOT it is exactly corrected, while if it occurred before it is restricted to weight one, while in the latter case the reverse holds. In either case, the only results are no overall error, or a single Z error on the current target register. This applies to all CNOT operations from the register block to the ancillas, excepting the final CNOT, which can propagate at most a single weight one error.

In this chapter we have presented methods for the generation of Clifford circuits for encoding, decoding and flagged fault tolerant measurement circuits for random codes. Next we will have a brief discussion of implementation details of this simulation before using the previously described error models and circuit simulation methods described in chapter 3 in conjunction with our circuits.

Chapter 5

Bit Bashing

Having discussed the operations required to perform Clifford gates on binary symplectic vectors, here we briefly address methods used for the storage and operations on our binary symplectic vectors and Kraus operators. These methods are essential for efficient computation as the size of our quantum system grows. For example; using binary values (as opposed to bytes or word sizes) to store binary vectors immediately decreases the memory requirements by a factor of 8 or 32 depending on the language used.

5.1 Unary and Binary Operations

Throughout this chapter we will be making reference to a number of unary and binary operations. Unless otherwise specified, all these operators act over binary strings.

- x `^` y: Bitwise XOR operator
- x `&` y: Bitwise AND operator
- x `&&` y: Bytewise AND operator acting over $\mathbb{Z}_{2^k}^n$ for some number of k bits.
- `~`x: Bitwise NOT operator
- `!`x : Bytewise NOT operator acting over $\mathbb{Z}_{2^k}^n$ for some number of k bits.
- x `<<` k and x `>>` k : Left and right bit shift operators. This treats the binary string as a vector and shifts each bit by k. Trailing elements are then set to zero.
- x `/` y: Division operator; notably this operator takes the floor of any non-integer components.
- x `%` y: Modulus operator
- x `?` y `:` z : Shorthand ternary switch using x as a truth statement, equivalent to : y if $x \neq 0$ else z.

Some types will also specify an explicit size in bits, for example `int8_t` is a signed integer type represented using two's compliment over 8 bits, while `uint64_t` is an unsigned integer over 64 bits.

5.2 Storage and Access on Binary Symplectic Vectors

An obvious approach to storing binary symplectic vectors is simply one bit per element. In most modern languages this poses a minor problem as the smallest accessible block size is eight bits, while the smallest allocatable block size is typically four bytes. While this can be easily overcome, it does force us to consider the bitwise operations required to allocate, store and access elements of our binary symplectic matrix.

We will begin by allocating a block large enough to store all the elements of our binary symplectic matrix. This is simply determining the number of bytes required to store the bits of the matrix. To reduce the runtime without elimintating readability, each of these operations will be performed using pre-processor macros rather than requiring calls to the function table.

```
// Calculate the number of bytes required to store the symplectic matrix
#define MATRIX_BYTES(s) {\
                        ((((s)->height) * ((s)->length)) % 8 ? \
                        ((((s)->height) * ((s)->length) ) / 8) + 1 : \
                        (((s)->height) * ((s)->length)) / 8) \
                }
```

Other approaches may store each row of the matrix in a new block, or even each X and Z component of each row in a new block. The latter reduces the calculation of the commutation relations to two XOR operations between the blocks, but as there is no single definition of the method of allocating memory within a struct, this approach would increase the difficulty of implementing our Hamming ordered generator later.

Having allocated our block, we now need to be able to access elements within the block. For this we calculate which byte the target bit lies in, and then find the relevant bit within the byte. Using these two values we can then construct and apply a bitmask to the relevant byte. Doubly negating the result of this bitmask returns the relevant bit to the rightmost position in the byte; a value of zero is perserved, while any value of one which is represented by a 1 in any position in the byte is mapped to 0 by the first NOT operation, and then to a 1 in the rightmost position by the second NOT operation.

```
/* Given a matrix, find the byte containing the i, j'th element
 * s is the symplectic matrix object
 * i specifies the row
 * j specifies the column
 */
#define BYTE_FROM_MATRIX(s, i, j) (((s)->length * (i) + (j)) / 8)

/* Given a matrix, find the bit field offset for the i, j'th element
 * s is the symplectic matrix object
 * i specifies the row
 * j specifies the column
 */
#define BIT_FROM_BYTE(s, i, j) (7 - ((s)->length * (i) + (j)) % 8)

/* Takes an element from a sym matrix
 * s is the symplectic matrix object
```

```
 * i specifies the row
 * j specifies the column
 */
#define ELEMENT_GET(s, i, j) {\
                        (!!((s)->matrix[BYTE_FROM_MATRIX(s, i, j)] & \
                        (1 << BIT_FROM_BYTE(s, i, j)))) \
                }
```

Similarly, we can use a bitmask and the same method to ensure that when updating the relevant bit, the rest of the byte is untouched. The bit is shifted the correct position and then a logical NOT operation is applied to the entire byte, forming a bitmask on all bits except the target. The new value can then be shifted to the correct position and a logical OR operation between it and the masked byte preserves the value of all the non-associated bits.

```
/* Stores an element in a sym matrix
 * s is the symplectic matrix object
 * i specifies the row
 * j specifies the column
 * v is the new value
 */
#define ELEMENT_SET(s, i, j, v) {\
                        ((s)->matrix[BYTE_FROM_MATRIX((s), (i), (j))] = \
                        ((s)->matrix[BYTE_FROM_MATRIX((s), (i), (j))] & \
                        (~(1 << BIT_FROM_BYTE((s), (i), (j))))) ^ (v) << \
                        BIT_FROM_BYTE((s), (i), (j))) \
                }
```

If the symplectic matrix object only consists of a single row we can then isolate the individual blocks with some simple shifts and masks.

```
uint64_t x_block = (uint64_t)s->matrix >> (s->length / 2);
uint64_t z_block = (uint64_t)s->matrix & ((1 << s->length / 2) - 1);
```

And the commutation relations can then be determined using a bitwise XOR and counting the number of set bits.

```
uint64_t commutation = x_block ^ z_block;
n_anticommuting = 0;
while (commutation != 0)
{
        commutation &= (commutation - 1);
        n_anticommuting++;
}
n_anticommuting %= 2;
```

This would normally be performed across two different symplectic matrix objects. Here subtracting 1 flips all bits up to and including the rightmost set bit. This with a logical and of itself unsets the right-most set bit. The number of times this operation is performed till all set bits are unset is then the number of set bits.

For symplectic vectors, matrix multiplication is simply xor within the same blocks.

```
x_block_a ^= x_block_b;
z_block_a ^= z_block_b;
```

In both of the above cases, acting on the symplectic matrices rather than just the vectors simply requires bit masking each row and then treating it as a vector.

5.3 Kraus Operator Representation

Each of the Kraus operators in the Pauli basis can be represented as a probability and a Pauli string. Typically this would require the storage of both the binary symplectic representation of the Pauli string and the associated probability; however using our single block representation of the binary symplectic matrix we can almost directly cast between our matrix and an integer representation.

```
uint64_t ll_rep = 0;
memcpy(s->matrix, &ll_rep, s->mem_size);
ll_rep >>= 64 - (s->length * s->height);
```

Our Kraus operator representation can then be constructed as a single array of floats, where the index of the element of the array can be cast to a binary symplectic vector as the Pauli string of the operator element. The value stored at that index is then the probability associated with the operator element. It's of note that this is limited to a 32 qubit register, or 64 element binary symplectic vector or matrix without moving to BIGNUM representation for the integers.

Updates on the Kraus operators can then be performed on the operation elements generated by the binary symplectic vectors and associated probabilities from the array elements.

5.4 Bill Gosper Hamming generator

The main advantage of this particular form of the symplectic vector is the ability to iterate through vectors in order of their X-Z Hamming weight. The Bill Gosper hamming weight transform takes a binary vector and returns the next ordered binary vector of the same Hamming weight. This allows us to construct a Hamming weight ordered generator without storing the full list of Kraus operators.

```
int64_t lowest_nonzero_bit = binary_vector & -binary_vector;
int64_t overflowed_vector = binary_vector + lowest_nonzero_bit;

binary_vector = (lowest_nonzero_bit != 0) ?
        (((overflowed_vector ^ binary_vector) >> 2) /
        lowest_nonzero_bit) | r : 0;
```

Using a similar approach to the commutation method above, under two's compliment taking the negative of a binary vector and performing a logical AND with itself returns just the lowest non-zero bit of the vector. When added back to the original vector this forces the lowest nonzero bit to flip to a zero and propagate a '1' upwards. This is functionally a left shift on the rightmost non-zero bit. It should be noted that this behaviour only occurs for signed types, and may fail for unsigned types.

Taking the XOR of that with the original val gives the set of bits that have been flipped by this operation, If the lowest set bit is nonzero then we can divide by it to eliminate the lowest set bit of the original vector. Following this with an OR operation with the original, this set of operations left shifts all bits on the left of our lowest nonzero bit until they reach another nonzero bit.

Using this generator we can now search over our integer indexed Kraus operator array by the Hamming weight of the X and Z elements in each Pauli string.

The generator can be used to iterate over a range between Hamming weights without needing to reach the end of the range simply by checking whether it has reached the last possible binary string for that particular Hamming weight (some number of 1's in the leftmost positions), and then increasing the Hamming weight and starting again from the first ordered Hamming weights (some number of 1's in the right most positions').

```
int64_t end_condition = ((1 << curr_hamming_weight) - 1)
        << (2 * curr_register_length - curr_hamming_weight);

int64_t next_vector = ((1 << curr_hamming_weight + 1) - 1);
```

We can also combine this generator with a bitmask and some shift operations to efficiently update on subsets of qubits in a register. Another use for this generator is in finding the lowest weighted destabilisers.

5.5 Expanding and Contracting the Register

Adding more qubits to the "end" of the register is a free operation; we now simply use additional indexes up to the new 2^{2n} limit. As qubits can be relabeled such that any new qubits are added to the "end" of the register this can be generalised to the expansion of the register always representing a free operation.

Conversely, removing elements from the register while preserving the probability distribution requires us to trace out all the indicies that are being removed. This is a much more expensive operation, and removing a single qubit requires iterating through three quarters of the Kraus operator array.

Chapter 6

Results

We have now developed a range of techniques for the construction, simulation and error minimisation of quantum circuits for small scale quantum devices. We can now consider deploying these techniques and developing a 'partial stack' compilation for a given quantum circuit on a characterised device. For a given logical gate or series of gates acting on the registers we can decompose this gate into a sequence of Clifford operations. This Clifford circuit will then depend upon the particular choice of encoded state.

For a set of error channels associated with the gates of a particular device, and for a selected error correcting code, we can exactly simulate the distribution of Pauli errors as the elements of the Clifford circuit are performed. By performing this simulation across a range of codes we can attempt to minimise the error rate of the circuit in respect to both logical gates being applied, and the error model of the device.

Armed with our random codes, generated Clifford circuits and techniques for efficiently updating our Kraus operators we can now start simulating and attempting to optimise circuits against particular choices of error models.

6.1 Simulation

All codes used for the simulations in this section are $[7, 1, 3]$ codes. Firstly we will consider the case of an ideal encoding circuit with some single round of biased IID Pauli noise applied. This can be seen in figure 6.1 and has been reproduced from [63]. Noticably in this figure there exist codes across the entire range of biases sampled that out-perform the Steane code. In addition a cyclic code has been constructed that closely follows the performance of the optimal random codes. This cyclic code will be used as part of our benchmark as we change the assumptions surrounding our error model.

We will consider the biased and IID error models described in section 3.3 and apply them both as an input to be corrected, and interleaved throughout the circuits.

Using the method for generating random codes outlined in chapter 3 and the Clifford circuit construction approach from chapter 4 we can find encoding circuits for arbitrary random codes. Using the error model from equation 3.6 for all preparation, measurement and single qubit Cliffords, in addition to the application of this model as an IID model for all participating qubits for multi-qubit gates and environmental noise, we can calculate the overall noise model of the encoding circuit.

From this noise model we can construct a tailored maximum likelihood decoder and determine the overall logical error rate of the encoding circuit.

6.2 Benchmark Codes

We will be looking at three fixed codes for the purposes of benchmarking (along with a range of random codes). The first is the Steane code [66], the stabilisers for which are a pair of classical Hamming codes, one measuring Z errors, the other X errors;

$$\begin{array}{ccccccc} Z & I & Z & I & Z & I & Z \\ I & Z & Z & I & I & Z & Z \\ I & I & I & Z & Z & Z & Z \\ X & I & X & I & X & I & X \\ I & X & X & I & I & X & X \\ I & I & I & X & X & X & X. \end{array}$$

In this particular representation the syndrome can be divided into an X block and a Z block, the binary representation of which indicates which qubit the error occurred on, forming a simple decoder.

The second is a seven qubit cyclic code that has previously demonstrated performance close to the best random codes for biased noise [63]. The stabilisers for this code are given by

$$\begin{array}{ccccccc} Z & X & I & I & X & Z & I \\ I & Z & X & I & I & X & Z \\ Z & I & Z & X & I & I & X \\ X & Z & I & Z & X & I & I \\ I & X & Z & I & Z & X & I \\ I & I & X & Z & I & Z & X. \end{array}$$

Finally the five qubit code, this is another cyclic code.

$$\begin{array}{ccccc} X & Z & I & Z & X \\ X & X & Z & I & Z \\ Z & X & X & Z & I \\ I & Z & X & X & Z. \end{array}$$

6.3 Simulation Results

As discussed previously, we can randomly sample the space of error correcting codes. It has been previously demonstrated, that for a particular choice of error model the performance of codes with the same distance and numbers of physical and logical qubits can vary. As a result, the overall performance of a circuit, can be improved by selecting a code appropriate for the error acting on the circuit.

Even for a single application of a depolarising or dephasing gate; we can see vast differences in the performance of these codes. This can be seen in figure 6.1. Here the probability of a logical error for different 7 qubit distance 3 stabiliser codes under a biased Pauli error model assuming ideal encoding, measurement and decoding operations. There is a clear demonstration that even for a simple error model, the overall error rate can be significantly reduced by selecting an appropriate error correcting code.

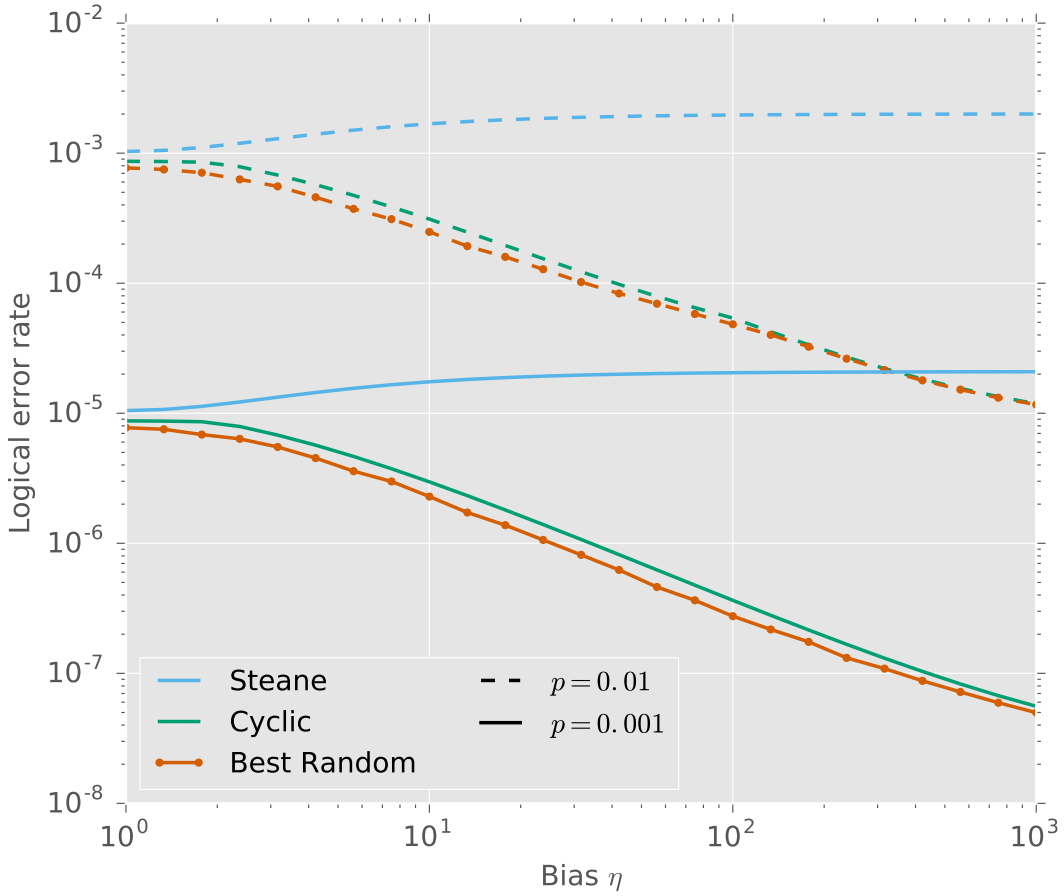

Figure 6.1: The probability of a logical error for different 7 qubit distance 3 stabiliser codes under a biased Pauli error model assuming ideal encoding, measurement and decoding operations. The blue line and green line denote the performance of the Steane code and the cyclic code respectively, while the orange line demonstrates the performance of the best random code for that error rate and bias[63]

We can make this simulation more realistic by expanding our model to include the overall

error channel produced by performing the encoding and decoding operations of the error correction procedure. This complicates the model further, first by applying multiple rounds of errors interleaved between the intended operations, by applying errors associated with the operations themselves and by multi-qubit operations causing the correlated spread of errors throughout the register.

Taking the error model from before we can now use the same error model as applied to each element of the circuit, and each non-participating wire as opposed to once before decoding. Applications of multi-qubit gates after applying this error model will act to spread errors and form correlated multi-qubit errors on our registers. We can perform this simulation for our encoding circuit with ideal measurement and recovery operations as shown in figure 6.2. In this figure we see the probability of a logical error for different 7 qubit distance 3 stabiliser codes under a biased Pauli error model assuming non-ideal encoding, but ideal measurement and decoding operations. Each Clifford gate in the encoding circuit has an associated Pauli error channel, each qubit that does not participate in a gate also has an error channel applied.

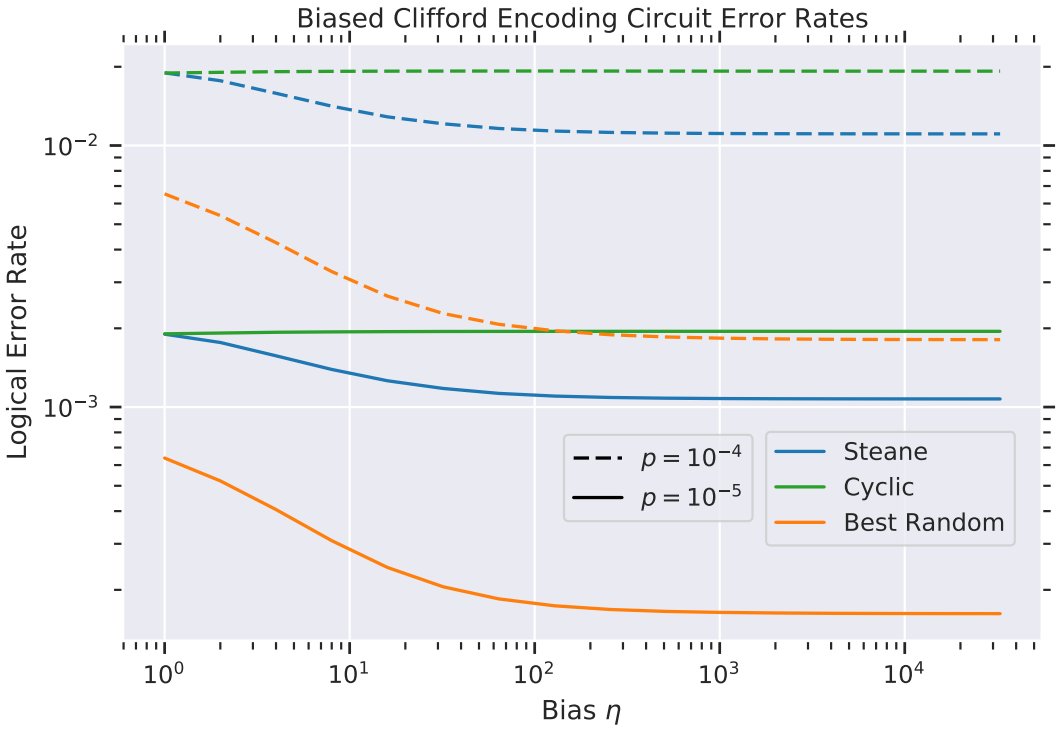

Figure 6.2: The probability of a logical error for different 7 qubit distance 3 stabiliser codes under a biased Pauli error model assuming non-ideal encoding, but ideal measurement and decoding operations. Each Clifford gate in the encoding circuit has an associated Pauli error channel, each qubit that does not participate in a gate also has an error channel applied. It can be seen that codes that perform or scale well in the single error channel model do not necessarily perform or scale well when repeated rounds of local Pauli error channels are applied. Sampling random codes in this case provides a significant advantage over more common codes. The encoding circuits used here are not fault tolerant.

It can be seen that the the comparative performance of codes under the assumption of a

noiseless encoding circuit not translate to the same comparative performance when circuit noise is simulated. In particular the cyclic code, which in the noiseless case closely tracks the best found random codes now performs significantly worse than even the Steane code. This demonstrates the necessity to tailor codes to the particular error model rather than searching using a single shot of the error gate.

By setting setting the wire errors and the gate errors to the same model we are searching for a tradeoff between the number of gates required to implement the encoding circuit, and the protection against errors provided by that code. It also demonstrates the significant advantage afforded by tailored random codes in these more complex error regimes and with larger and more varied range of circuits.

It is worth spending a moment discussing the idea of a physical error rate in this context. Unlike the earlier single shot model where the error channel is applied once (as seen in figure 3.3) we are now interleaving our errors through gates (figure 3.4). The reported physical error rate is the error rate of *each* of these interleaved gates. For our typical 7 qubit encoding circuit constructed using only the Clifford generators, and with all gates separated such that we cannot perform more than one gate at a time, we typically see a range of 40 to 80 gates, as seen in figure 6.3.

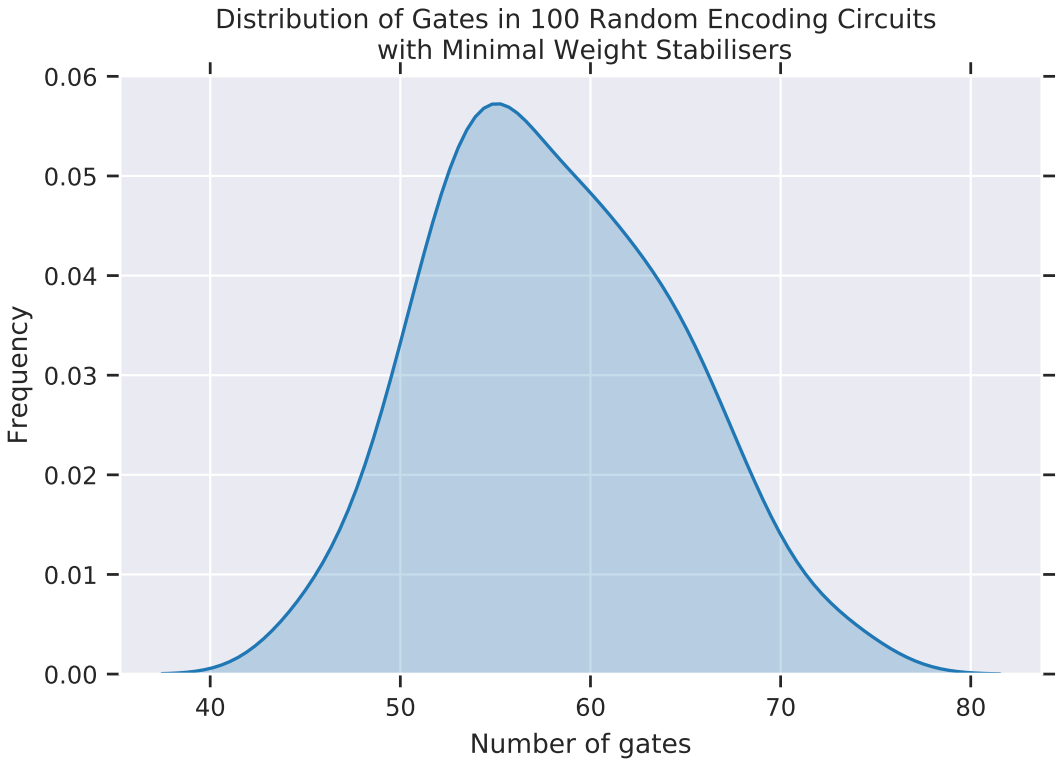

Figure 6.3: The distribution of the number of Clifford gates are contained within each of 100 random encoding circuits with minimum weight stabilisers for the encoding circuit generation method discussed in chapter 4.

This distribution complicates the relationship between the physical error rate and the logical error rate. As can be see in figures for the circuits in this chapter, the physical error rate is much

lower than the logical error rate. The reported physical error rate is the rate for each of the interleaved error channels, while the logical error rate is the error rate of the circuit as a whole. This process is further complicated when we consider that multi qubit gates (such as CNOTs) permit errors to increase in weight as they propagate throughout the circuit.

We can extend the previous simulation from the encoding and decoding circuits to also include measurement. This significantly increases the computational overhead of the simulation as we now need to also track the state of a range of ancilla qubits. Furthermore we need to include a notion of measurement for these qubits that does not naturally fall within the established state independent Heisenberg interpretation. The handling of these complexities have been dealt with in previous chapters, but it is a noticeable point of distinction between measurement circuits and circuits that only perform logical operations or encoding.

Unlike encoding circuits, there is much more uniformity across the measurement circuits. Most stabiliser measurement circuits are near identical excepting the weight of the stabilisers and the number of basis transformations required. As can be seen in figure 6.4 there is only a marginal gain in the random codes compared to the Steane and cyclic codes. Here we apply the same techniques and make the same assumptions as the encoding circuit with a minor change; we see the probability of a logical error for different 7 qubit distance 3 stabiliser codes under a biased Pauli error model assuming ideal encoding, but non-ideal measurement operations.

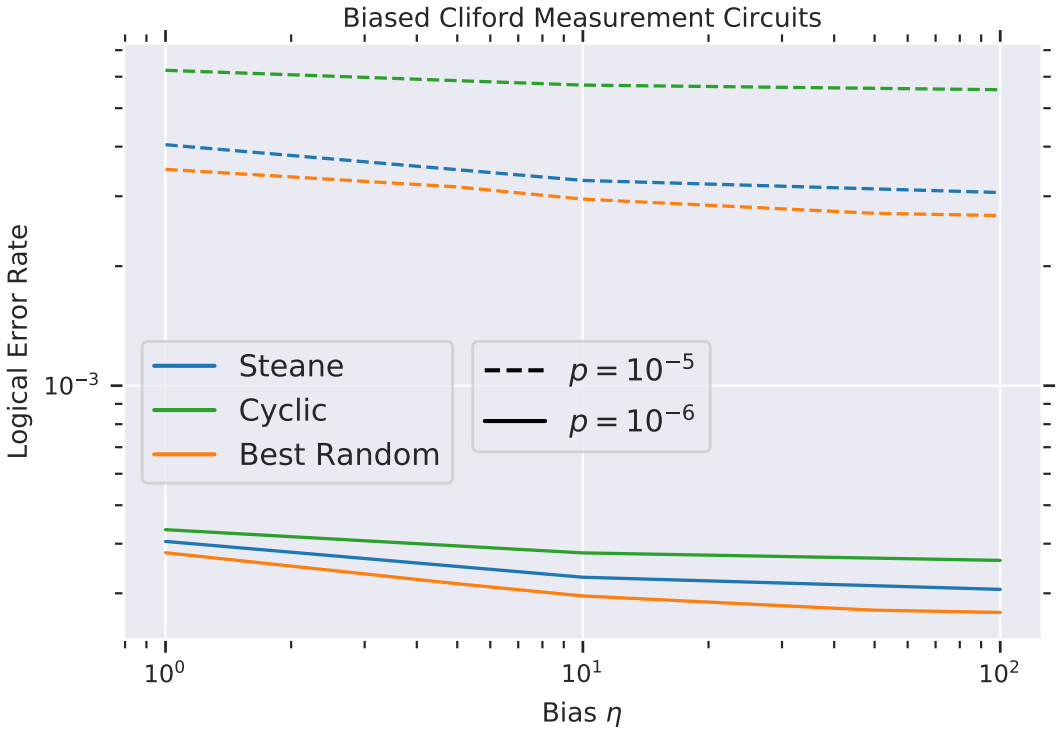

Figure 6.4: The probability of a logical error for different 7 qubit distance 3 stabiliser codes under a biased Pauli error model assuming ideal encoding and decoding circuits, but non-ideal measurement operations. The blue line and green line denote the performance of the Steane code and the cyclic code respectively, while the orange line demonstrates the performance of the best random code for that error rate and bias.

We can extend our measurement circuits using flagged fault tolerance, as can be seen in figure 6.5. Adding flag qubits alongside our ancilla qubits from the measurement circuit continues to increase the memory and time requirements for simulation, however we can now partially suppress the spread of errors in multi-qubit gates. Ultimately, if the physical error rate is low enough, the use of fault tolerant gates will allow us to build scalable concatenated codes and eventually reduce the logical error rate to near zero.

As before, we see the probability of a logical error for different 7 qubit distance 3 stabiliser codes under a biased Pauli error model assuming non-ideal encoding, but ideal measurement and decoding operations. We also assume that our flag qubits are non-ideal and are otherwise identical to our ancilla and register qubits; they explicitly do not have a lower associated error rate.

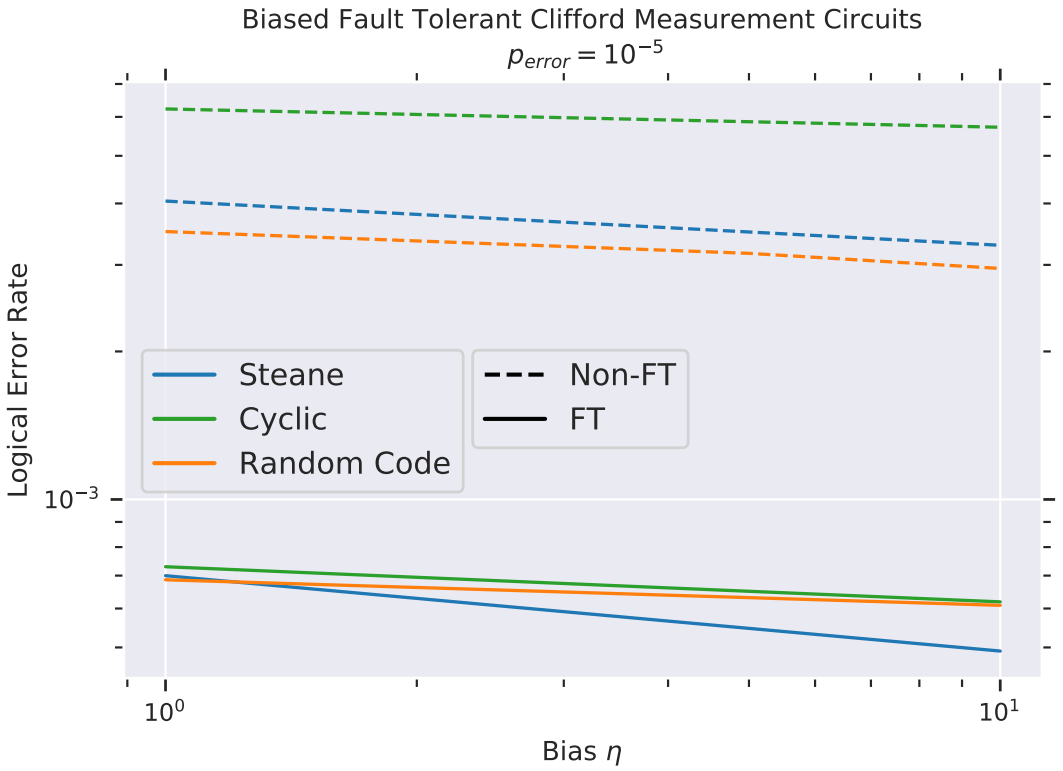

Figure 6.5: The probability of a logical error for different 7 qubit distance 3 stabiliser codes under a biased Pauli error model assuming ideal encoding and decoding circuits, but non-ideal fault tolerant measurement operations compared to non-fault tolerant measurement operations. The blue line and green line denote the performance of the Steane code and the cyclic code respectively, while the orange line demonstrates the performance of the best random code for that error rate and bias.

Incorporating fault tolerance into these circuits has reduced the overall error rate across all the codes compared, but does not necessarily preserve the performance of the codes relative to each other. Again, this is perhaps due to the high degrees of similarity between the circuits required for measurement and flagged fault tolerance, when compared to the large range of circuits seen during encoding and decoding. More importantly, we begin to see advantages from

incorporating fault tolerance even while the logical error rate is still greater than the physical error rate.

While we are considering these improvements, we will also take a brief look into some average rates of improvement for a non-fault tolerant encoding circuit with a range of different error models. We can see that relatively large gains can be made on average even a within the first few samples of random codes.

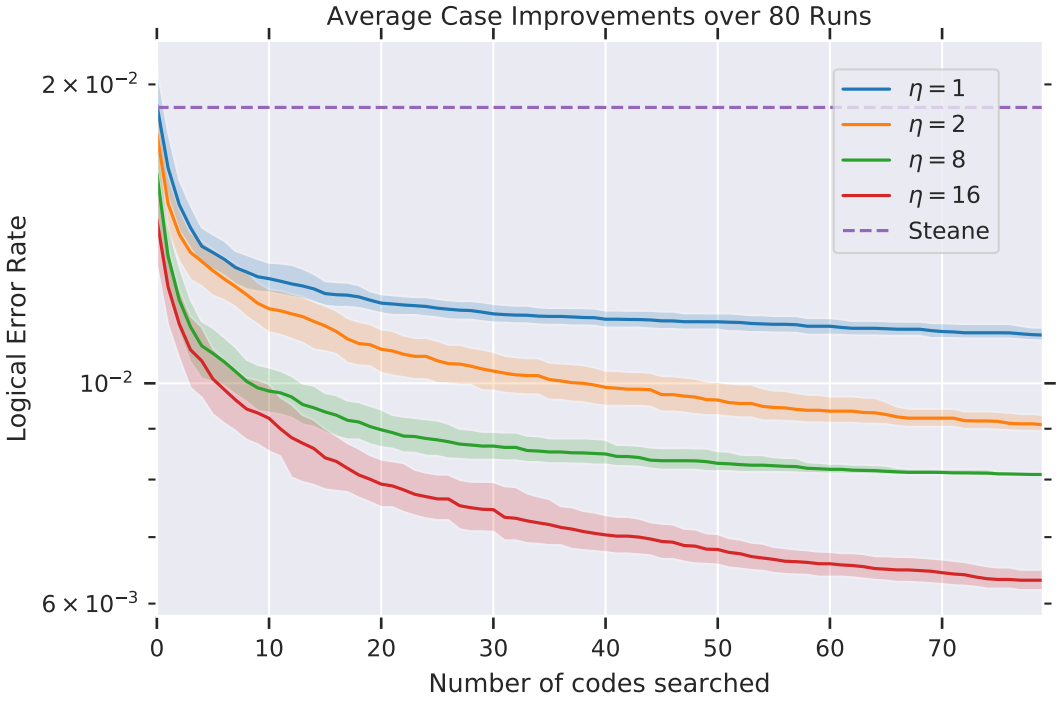

Figure 6.6: The performance of the average best code for a range of sampled encoding circuits over a range of biases. In this particular circumstance it can be seen that a relatively good code can be found within a few samples. The figure here represents the average of a 100 iterations of tracking current best code for 80 samples of the random codes. Here the Steane code has been plotted for $\eta = 1$.

More importantly, we can see that in the average case we instantly improve upon the Steane code with a random code, and that we make further improvements within the first 10 random codes we search. This indicates that even a relatively small amount of pre-processing yields relatively significant gains for random encoding circuits.

As for the circuits themselves, while the syndrome measurement circuit is fault tolerant (see appendix A), we can see that errors introduced during the syndrome measurement can produce incorrect syndromes. Our decoder then reads the incorrect syndrome and applies an incorrect recovery operation, introducing the possibility of an incorrect logical recovery.

In figure 6.7, we can see two rounds of syndrome measurement and recovery; the first with noisy Hadamard gates, perfect state preparation, measurement and wire noise, the second with perfect Hadamards as before.

As the gate error rate decreases, the bound on the second, perfect syndrome measurement and recovery circuits is set by the gate error rate, indicating that a single gate error has been mapped

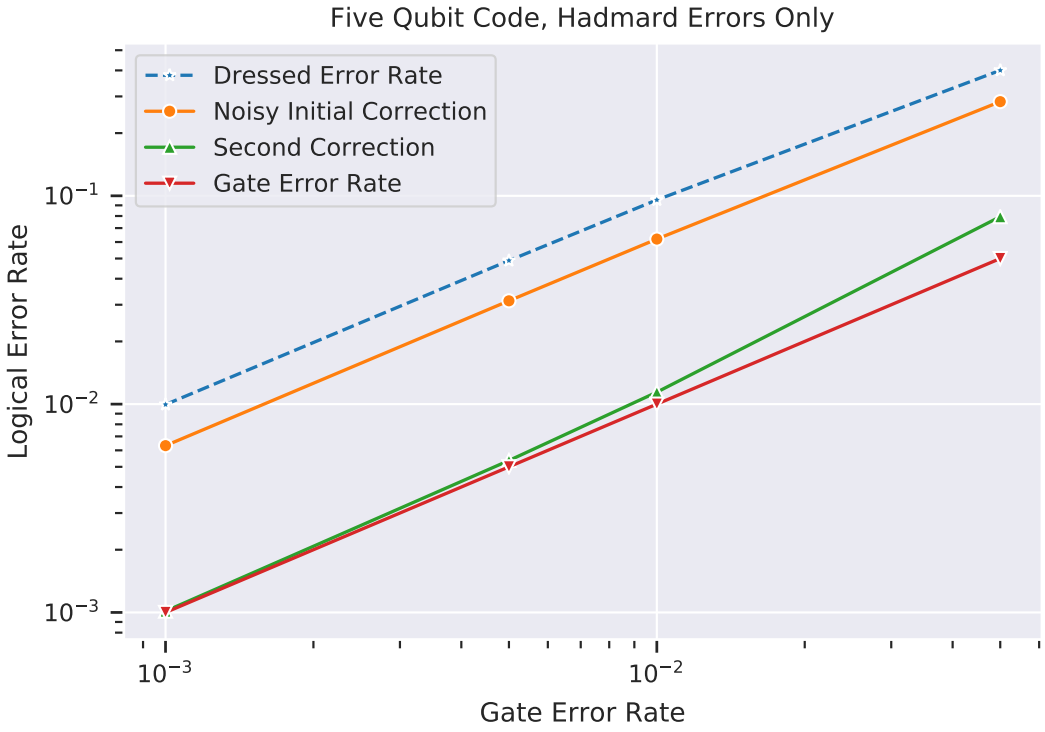

Figure 6.7: The five qubit code with IID errors only on Hadmard gates. The dressed error rate is the error rate of the circuit prior to any recovery operations taking place, as opposed to the gate error rate which is the error rate of each gate. The syndrome measurement and recovery circuits are run twice, the first time with the error model applied, while the second is run with no errors. We can see that the perfect second correction is bounded by a logical error produced during the decoding of the first round. This is made more apparent when compared to figure 6.8, where the same initial noisy measurement occurs, but no recovery operation takes place during the first round.

to an incorrect logical state in the first round. This can be compared against figure 6.8, where the same initially noisy syndrome measurement circuit is applied, but the syndromes are discarded and no first round of recovery operations occurs before the noiseless circuit is applied. These two figures indicate that there exists a single site in the five qubit syndrome measurement circuit that produces an incorrect syndrome resulting in an overall logical error. It is suggested, that it might be possible to procedurally detect and flag these individual locations using a single additional ancilla qubit performing parity checks across each bad location using stabiliser measurements. As with flagged fault tolerance, when the ancilla is tripped, the measured syndromes would be discarded, and the syndrome measurement restarted. Figure 6.8 exhibits a threshold at a gate error rate of approximately 2%, where the logical error rate after correction has been sufficiently suppressed as to be less than the gate error rate.

Furthermore, the error rate will be bounded by the total probability of wire or gate errors on the final set of gates in the syndrome measurement. At this point in the circuit all syndromes have been measured, and any errors in the register qubits are undetectable. This is equivalent to the case in figure 6.9.

Figure 6.8: The five qubit code with IID errors only on Hadmard gates. The dressed error rate is the error rate of the circuit prior to any recovery operations taking place, as opposed to the gate error rate which is the error rate of each gate. The syndrome measurement circuit is run twice, while the recovery circuit is only run at the very end. The first run of the syndrome measurement occurs with the error model applied, while the second is run with no errors. We can see that the perfect second correction exhibits a threshold and surpasses the gate error rate. This is made more apparent when compared to figure 6.7, where the same initial noisy measurement occurs, and a noisy recovery operation takes place during the first round.

We can also perform multiple rounds of noisy fault tolerant syndrome measurement and recovery prior to a noiseless round. Using the same IID error model as before, applied only to Hadamard gates in the circuit we can see this behaviour after nine rounds of noisy measurement in figure 6.10.

In this section we have demonstrated an approach to error aware compilation of quantum encoding circuits, with a particular focus on Clifford circuits for encoding, measurement and flagged fault tolerance. These techniques may be generalised to all Clifford circuits for the purposes of tracking and minimising error channels for small numbers of qubits.

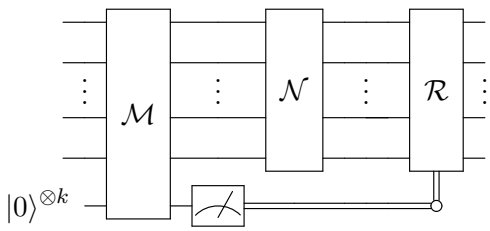

Figure 6.9: Errors occurring on the final gates of the syndrome measurement circuit act as undetected errors. They are equivalent to errors taking place between the measurement and recovery operations.

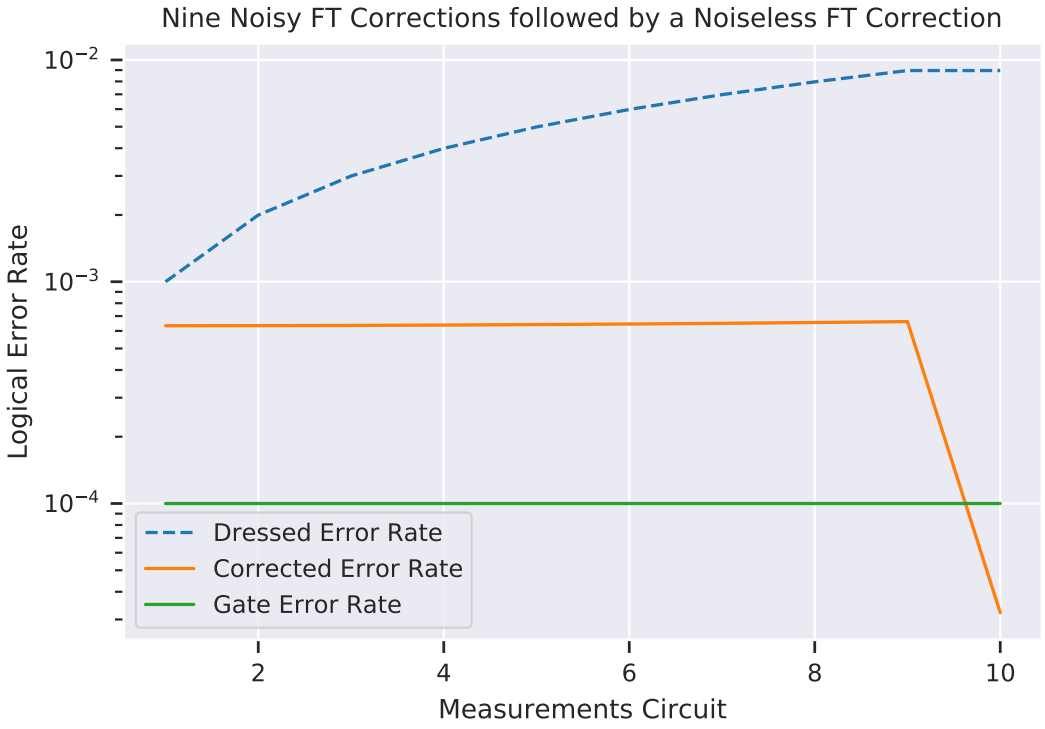

Figure 6.10: The five qubit code with IID errors only on Hadmard gates. The dressed error rate is the probability with which any error has occurred. The syndrome measurement and noisy recovery circuits are run nine times prior to a noiseless recovery circuit at the end. We can see that the perfect final correction exhibits a threshold and surpasses the gate error rate.

Chapter 7

Future Work and Conclusion

In this thesis we presented a scheme for incorporating information about the error model into a Clifford compilation with error correction. This represents a change of approach in current circuit optimisation methods; from heuristic approaches such as T gate and CNOT gate counts to error aware modeling and circuit construction. It also extends on most treatments of correlated errors; considering an arbitrary probability distribution over the set of Kraus operators.

This scheme has demonstrated marked improvements in the logical error rate of an error correcting code applied over some number of physical qubits. As this logical error rate decrease, the lifetime of the system increases and the number of gate operations that can be performed in a circuit increase.

Having presented a tool for investigating optimal codes within a given error model, we can extend the use of the tool for more realistic error models. While the Hadamard model on five qubits exhibits a threshold, the error model in use can be modified to incorporate current quantum systems.

The current simulation, while classically efficient, does not incorporate a model of universal quantum computation. We plan to incorporate T gates into the simulation and extend our model from Clifford circuits to universal quantum computation. While this would significantly increase the computational overhead of the simulation, expanding from the Clifford gates permits a far greater range of operations. This will likely be aided by recent advances in the simulation of T gates such as [8]. By expanding the simulation of universal quantum computation, we can now attempt to compile and optimise arbitrary quantum gates.

One important consideration for any additional circuits or gates that would be generated by our scheme is Fault Tolerance. Another natural extension of this work is to implement FT compilation for the encoding circuits in addition to the current flag fault tolerant measurement circuits. This can be hopefully extended to finding error aware implementations of fault tolerant gates for universal quantum computation. Incorporating fault tolerant gates with a sufficiently suppressed error rate permits the implementation of concatenated codes. With a sufficient number of physical qubits, these concatenated codes would reduce the error rate to effectively 0; greatly increasing the number of gates that can be performed before the system decoheres. Additional features can be added to further suppress errors, such as implementing overcomplete syndrome measurements or repeated syndrome measurements, at the cost of an increased computational overhead.

'Efficiency' is a somewhat open ended goal, and again, with multiple relevant figures of merit. Classically we can consider reducing the number of operations required to implement a particular gate (often by increasing the number of elements in the gate set), reducing the overall runtime of the program or reducing the number of elements in the gate set and hence the implementation

difficulty. For our purposes, we will consider our figure of merit to be the error rate of the implementation of the operation.

Lastly, this approach can be benchmarked using the data from error models from real devices around the 16 qubit range. Using this error characterisation can expand these techniques into a dedicated compiler pipeline for a particular device. This would comprise of a section of a quantum compilation stack for the device and would play a part in the development of 'full stack' quantum computing.

Bibliography

[1] Scott Aaronson and Daniel Gottesman. "Improved simulation of stabilizer circuits". In: *Phys. Rev. A* 70 (5 2004), p. 052328. DOI: 10.1103/PhysRevA.70.052328. URL: https://link.aps.org/doi/10.1103/PhysRevA.70.052328.

[2] Dorit Aharonov. "A Simple Proof that Toffoli and Hadamard are Quantum Universal". In: *arXiv:0301040 [quant-ph]* (Jan. 2003). arXiv: 0301040.

[3] Gadi Aleksandrowicz and Thomas Alexander et al. *Qiskit: An Open-source Framework for Quantum Computing*. 2019. DOI: 10.5281/zenodo.2562110.

[4] P Aliferis et al. "Fault-tolerant computing with biased-noise superconducting qubits: a case study". In: *New Journal of Physics* 11.1 (2009), p. 013061. DOI: 10.1088/1367-2630/11/1/013061. URL: http://stacks.iop.org/1367-2630/11/i=1/a=013061.

[5] Panos Aliferis and John Preskill. "Fault-tolerant quantum computation against biased noise". In: *Phys. Rev. A* 78 (5 2008), p. 052331. DOI: 10.1103/PhysRevA.78.052331. URL: http://link.aps.org/doi/10.1103/PhysRevA.78.052331.

[6] Harrison Ball et al. "The Effect of Noise Correlations on Randomized Benchmarking". In: *Phys. Rev. A* 93 (2016), p. 022303. DOI: 10.1103/PhysRevA.93.022303. arXiv: 1504.05307.

[7] Adriano Barenco et al. "Elementary gates for quantum computation". In: *Phys. Rev. A* 52 (5 1995), pp. 3457–3467. DOI: 10.1103/PhysRevA.52.3457. URL: https://link.aps.org/doi/10.1103/PhysRevA.52.3457.

[8] Sergey Bravyi and David Gosset. "Improved Classical Simulation of Quantum Circuits Dominated by Clifford Gates". In: *Phys. Rev. Lett.* 116 (25 2016), p. 250501. DOI: 10.1103/PhysRevLett.116.250501. URL: https://link.aps.org/doi/10.1103/PhysRevLett.116.250501.

[9] Sergey Bravyi and Alexei Kitaev. "Universal quantum computation with ideal Clifford gates and noisy ancillas". In: *Phys. Rev. A* 71 (2 2005), p. 022316. DOI: 10.1103/PhysRevA.71.022316. URL: https://link.aps.org/doi/10.1103/PhysRevA.71.022316.

[10] Peter Brooks and John Preskill. "Fault-Tolerant Quantum Computation with Asymmetric Bacon-Shor Codes". In: *Physical Review A* 87.3 (Mar. 2013). ISSN: 1050-2947, 1094-1622. DOI: 10.1103/PhysRevA.87.032310. arXiv: 1211.1400.

[11] A. R. Calderbank and Peter W. Shor. "Good quantum error-correcting codes exist". In: *Phys. Rev. A* 54 (2 1996), pp. 1098–1105. DOI: 10.1103/PhysRevA.54.1098. URL: http://link.aps.org/doi/10.1103/PhysRevA.54.1098.

[12] Christopher Chamberland and Michael E. Beverland. "Flag fault-tolerant error correction with arbitrary distance codes". In: *Quantum* 2 (Feb. 2018), p. 53. ISSN: 2521-327X. DOI: 10.22331/q-2018-02-08-53. URL: https://doi.org/10.22331/q-2018-02-08-53.

[13] Joshua Combes et al. "In-Situ Characterization of Quantum Devices with Error Correction". In: (May 2014). arXiv: 1405.5656 [quant-ph].

[14] A. D. Córcoles et al. "Demonstration of a Quantum Error Detection Code Using a Square Lattice of Four Superconducting Qubits". en. In: *Nature Communications* 6 (Apr. 2015), p. 6979. ISSN: 2041-1723. DOI: 10.1038/ncomms7979.

[15] A. D. Córcoles et al. "Process Verification of Two-Qubit Quantum Gates by Randomized Benchmarking". In: *Physical Review A* 87.3 (Mar. 2013), p. 030301. DOI: 10.1103/PhysRevA.87.030301.

[16] D. G. Cory et al. "Experimental Quantum Error Correction". In: *Physical Review Letters* 81.10 (Sept. 1998), pp. 2152–2155. DOI: 10.1103/PhysRevLett.81.2152.

[17] Ben Criger, Osama Moussa, and Raymond Laflamme. "Study of Multiple Rounds of Error Correction in Solid State NMR QIP". In: (Mar. 2011). arXiv: 1103.4396 [quant-ph].

[18] Andrew W. Cross et al. "Open Quantum Assembly Language". In: *arXiv:1707.03429 [quant-ph]* (July 2017). arXiv: 1707.03429. URL: http://arxiv.org/abs/1707.03429.

[19] S. D. Bartlett M. W. Mitchell G. J. Pryde D. W. Berry B. L. Higgins and H. M. Wiseman. "How to perform the most accurate possible phase measurements". In: *Phys. Rev.A* (2009). DOI: 10.1103/PhysRevA.80.052114.

[20] S. Debnath et al. "Demonstration of a Small Programmable Quantum Computer with Atomic Qubits". en. In: *Nature* 536.7614 (Aug. 2016), pp. 63–66. ISSN: 0028-0836. DOI: 10.1038/nature18648.

[21] N. Delfosse and J. P. Tillich. "A decoding algorithm for CSS codes using the X/Z correlations". In: *2014 IEEE International Symposium on Information Theory*. 2014, pp. 1071–1075. DOI: 10.1109/ISIT.2014.6874997.

[22] D. Deutsch. "Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer". In: *Proc. R. Soc. Lond.* (1985).

[23] David Deutsch and Richard Jozsa. "Rapid Solution of Problems by Quantum Computation". en. In: *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 439.1907 (Dec. 1992), pp. 553–558. ISSN: 1364-5021, 1471-2946. DOI: 10.1098/rspa.1992.0167.

[24] Guillaume Duclos-Cianci and David Poulin. "Reducing the quantum-computing overhead with complex gate distillation". In: *Phys. Rev. A* 91 (4 2015), p. 042315. DOI: 10.1103/PhysRevA.91.042315. URL: http://link.aps.org/doi/10.1103/PhysRevA.91.042315.

[25] Joseph Emerson, Robert Alicki, and Karol Zyczkowski. "Scalable Noise Estimation with Random Unitary Operators". In: *Journal of Optics B: Quantum and Semiclassical Optics* 7.10 (2005), S347–S352. ISSN: 1464-4266.

[26] Jeffrey M. Epstein et al. "Investigating the limits of randomized benchmarking protocols". In: *Phys. Rev. A* 89 (6 2014), p. 062321. DOI: 10.1103/PhysRevA.89.062321. URL: http://link.aps.org/doi/10.1103/PhysRevA.89.062321.

[27] I. Kassal et.al. "Polynomial-time quantum algorithm for the simulation of chemical dynamics". In: *Proceedings of the National Academy of Sciences of the United States of America* 105.48 (). DOI: 10.1073/pnas.0808245105.

[28] R.P. Feynman. "Simulating Physics with Computers". In: *Int. J. Theor. Phys.* 21 (1982), pp. 467–488.

[29] Jan Florjanczyk and Todd A. Brun. "In-Situ Adaptive Encoding for Asymmetric Quantum Error Correcting Codes". In: (Dec. 2016). arXiv: 1612.05823 [quant-ph].

[30] M. A. Fogarty et al. "Nonexponential fidelity decay in randomized benchmarking with low-frequency noise". In: *Phys. Rev. A* 92 (2 2015), p. 022326. DOI: 10.1103/PhysRevA.92.022326. URL: http://link.aps.org/doi/10.1103/PhysRevA.92.022326.

[31] Austin G. Fowler et al. "Scalable Extraction of Error Models from the Output of Error Detection Circuits". In: (May 2014). arXiv: 1405.1454 [quant-ph].

[32] Austin G. Fowler et al. "Surface codes: Towards practical large-scale quantum computation". In: *Phys. Rev. A* 86 (3 2012), p. 032324. DOI: 10.1103/PhysRevA.86.032324. URL: http://link.aps.org/doi/10.1103/PhysRevA.86.032324.

[33] Steven M. Girvin. *Superconducting Qubits and Circuits: Artificial Atoms Coupled to Microwave Photons*. 2011.

[34] Daniel Gottesman. "An Introduction to Quantum Error Correction and Fault-Tolerant Quantum Computation". In: *quant-ph* (). URL: arXiv:0904.2557v1.

[35] Daniel Gottesman. "Fault-Tolerant Quantum Computation with Constant Overhead". In: (Oct. 2013). arXiv: 1310.2984 [quant-ph].

[36] Daniel Gottesman. "The Heisenberg Representation of Quantum Computers". In: *arXiv:9807006 [quant-ph]* (July 1998). arXiv: 9807006.

[37] Lov K. Grover. "A Fast Quantum Mechanical Algorithm for Database Search". In: *arXiv:quant-ph/9605043* (May 1996). arXiv: quant-ph/9605043.

[38] Mauricio Gutiérrez and Kenneth R. Brown. "Comparison of a Quantum Error Correction Threshold for Exact and Approximate Errors". In: *Physical Review A* 91.2 (Feb. 2015), p. 022335. ISSN: 1050-2947, 1094-1622. DOI: 10.1103/PhysRevA.91.022335. arXiv: 1501.00068.

[39] Robin Harper and Steven T. Flammia. "Fault-Tolerant Logical Gates in the IBM Quantum Experience". In: *Physical Review Letters* 122.8 (Feb. 2019), p. 080504. DOI: 10.1103/PhysRevLett.122.080504. URL: https://link.aps.org/doi/10.1103/PhysRevLett.122.080504.

[40] James Humphreys. *Introduction to Lie Algebras and Representation Theory*. Springer, 1997.

[41] Tomas Jochym-O'Connor et al. "The Robustness of Magic State Distillation against Errors in Clifford Gates". In: (May 2012). arXiv: 1205.6715 [quant-ph].

[42] Cody Jones. "Multilevel distillation of magic states for quantum computing". In: *Phys. Rev. A* 87 (4 2013), p. 042305. DOI: 10.1103/PhysRevA.87.042305. URL: http://link.aps.org/doi/10.1103/PhysRevA.87.042305.

[43] Jupyter Development Team. *Jupyter*. Aug. 2016. URL: https://github.com/jupyter/jupyter.

[44] Ivan Kassal et al. "Polynomial-Time Quantum Algorithm for the Simulation of Chemical Dynamics". en. In: *Proceedings of the National Academy of Sciences* 105.48 (Feb. 2008), pp. 18681–18686. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.0808245105.

[45] J. Kelly et al. "State Preservation by Repetitive Error Detection in a Superconducting Quantum Circuit". en. In: *Nature* 519.7541 (Mar. 2015), pp. 66–69. ISSN: 0028-0836. DOI: 10.1038/nature14270.

[46] N. Khammassi et al. "cQASM v1.0: Towards a Common Quantum Assembly Language". In: *arXiv:1805.09607 [quant-ph]* (May 2018). arXiv: 1805.09607. URL: http://arxiv.org/abs/1805.09607.

[47] E. Knill. "Quantum Computing with Realistically Noisy Devices". In: *Nature* 434.7029 (Mar. 2005), pp. 39–44. ISSN: 0028-0836. DOI: 10.1038/nature03350.

[48] Muyuan Li et al. "Fault tolerance with bare ancillary qubits for a [[7,1,3]] code". In: *Phys. Rev. A* 96 (3 2017), p. 032341. DOI: 10.1103/PhysRevA.96.032341. URL: https://link.aps.org/doi/10.1103/PhysRevA.96.032341.

[49] N. M. Linke et al. "Experimental demonstration of quantum fault tolerance". In: (Nov. 2016). arXiv: 1611.06946 [quant-ph].

[50] Adam M. Meier, Bryan Eastin, and Emanuel Knill. "Magic-State Distillation with the Four-Qubit Code". In: (Apr. 2012). arXiv: 1204.4221 [quant-ph].

[51] Osama Moussa et al. "Demonstration of Sufficient Control for Two Rounds of Quantum Error Correction in a Solid State Ensemble Quantum Information Processor". In: *Physical Review Letters* 107.16 (Oct. 2011), p. 160501. DOI: 10.1103/PhysRevLett.107.160501.

[52] M.A. Nielsen and I.L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000. DOI: 10.1017/CBO9780511976667.

[53] D. Nigg et al. "Quantum computations on a topologically encoded qubit". In: *Science* 345.6194 (2014), pp. 302–305. ISSN: 0036-8075. DOI: 10.1126/science.1253742. URL: http://science.sciencemag.org/content/345/6194/302.

[54] Shin Nishio et al. "Extracting Success from IBM's 20-Qubit Machines Using Error-Aware Compilation". In: *arXiv:1903.10963 [quant-ph]* (Mar. 2019). arXiv: 1903.10963. URL: http://arxiv.org/abs/1903.10963.

[55] Harold Ollivier and Jean-Pierre Tillich. "Description of a Quantum Convolutional Code". In: *Phys. Rev. Lett.* 91 (17 2003), p. 177902. DOI: 10.1103/PhysRevLett.91.177902. arXiv: quant-ph/0304189.

[56] Alexander Pitchford et al. *QuTiP 4.1.0*. 2016–. URL: https://github.com/qutip/qutip.

[57] John Preskill. "Fault-Tolerant Quantum Computation". In: (Dec. 1997). arXiv: quant-ph/9712048.

[58] John Preskill. "Quantum Computing in the NISQ era and beyond". In: *Quantum* 2 (Aug. 2018), p. 79. ISSN: 2521-327X. DOI: 10.22331/q-2018-08-06-79. URL: https://doi.org/10.22331/q-2018-08-06-79.

[59] Daniel Puzzuoli et al. "Tractable Simulation of Error Correction with Honest Approximations to Realistic Fault Models". In: *Physical Review A* 89.2 (Feb. 2014), p. 022306. DOI: 10.1103/PhysRevA.89.022306.

[60] R Raussendorf, J Harrington, and K Goyal. "Topological fault-tolerance in cluster state quantum computation". In: *New Journal of Physics* 9.6 (2007), p. 199. DOI: 10.1088/1367-2630/9/6/199. URL: http://stacks.iop.org/1367-2630/9/i=6/a=199.

[61] Robert Raussendorf, Jim Harrington, and Kovid Goyal. "Topological Fault-Tolerance in Cluster State Quantum Computation". In: *New Journal of Physics* 9.6 (June 2007), pp. 199–199. ISSN: 1367-2630. DOI: 10.1088/1367-2630/9/6/199. arXiv: quant-ph/0703143.

[62] M. D. Reed et al. "Realization of Three-Qubit Quantum Error Correction with Superconducting Circuits". en. In: *Nature* 482.7385 (Feb. 2012), pp. 382–385. ISSN: 0028-0836. DOI: 10.1038/nature10786.

[63] Alan Robertson et al. "Tailored Codes for Small Quantum Memories". In: *Phys. Rev. Applied* 8 (6 2017), p. 064004. DOI: 10.1103/PhysRevApplied.8.064004. URL: https://link.aps.org/doi/10.1103/PhysRevApplied.8.064004.

[64] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: (Aug. 1995). arXiv: quant-ph/9508027.

[65] M. D. Shulman et al. "Demonstration of Entanglement of Electrostatically Coupled Singlet-Triplet Qubits". In: *Science* 336.6078 (2012), pp. 202–205. ISSN: 0036-8075. DOI: 10.1126/science.1217692. URL: http://science.sciencemag.org/content/336/6078/202.

[66] A. M. Steane. "Error Correcting Codes in Quantum Theory". In: *Phys. Rev. Lett.* 77.5 (1996), pp. 793–797. DOI: 10.1103/PhysRevLett.77.793.

[67] Markku P. V. Stenberg, Oliver Köhn, and Frank K. Wilhelm. "Characterization of Decohering Quantum Systems: Machine Learning Approach". In: *Physical Review A* 93.1 (Jan. 2016). ISSN: 2469-9926, 2469-9934. DOI: 10.1103/PhysRevA.93.012122. arXiv: 1510.05655.

[68] David K. Tuckett, Stephen D. Bartlett, and Steven T. Flammia. "Ultrahigh Error Threshold for Surface Codes with Biased Noise". In: *Phys. Rev. Lett.* 120 (5 2018), p. 050505. DOI: 10.1103/PhysRevLett.120.050505. URL: https://link.aps.org/doi/10.1103/PhysRevLett.120.050505.

[69] A. M. Turing. "On Computable Numbers with an Application to the Entscheidungsproblem". In: *Proceedings of the London Mathematical Society* 42.2 (1936).

[70] M. Veldhorst et al. "A Two-Qubit Logic Gate in Silicon". en. In: *Nature* 526.7573 (Oct. 2015), pp. 410–414. ISSN: 0028-0836. DOI: 10.1038/nature15263.

[71] Paul Webster, Stephen D. Bartlett, and David Poulin. "Reducing the overhead for quantum computation when noise is biased". In: *Phys. Rev. A* 92 (6 2015), p. 062309. DOI: 10.1103/PhysRevA.92.062309. URL: http://link.aps.org/doi/10.1103/PhysRevA.92.062309.

[72] Xiao-Gang Wen. "Quantum Orders in an Exact Soluble Model". In: *Phys. Rev. Lett.* 90 (1 2003), p. 016803. DOI: 10.1103/PhysRevLett.90.016803. URL: http://link.aps.org/doi/10.1103/PhysRevLett.90.016803.

[73] W.H. Zurek W.K. Wooters. "A single quantum cannot be cloned". In: 299 (1982), pp. 802–803.

[74] Yanqing Wu et al. "Quantum Behavior of Graphene Transistors near the Scaling Limit". In: *Nano Letters* 12.3 (Mar. 2012), pp. 1417–1423. ISSN: 1530-6984. DOI: 10.1021/nl204088b.

Appendix A

A.1 Demonstration of Flagged Fault Tolerance on 5 Qubits

The following section details the application of a Z error on the ancilla qubit at each point in the volume of the syndrome measurement circuit. After correction using the flag qubits, the error is of at most weight one modulo the stabiliser group demonstrating that the circuit is fault tolerant. As the flag qubits were tripped, the syndrome measurement is repeated (this time with no errors) and the error is corrected. It can be seen that in each instance the state is returned to the code-space with no overall logical error.

The code for performing this check may be found at https://github.com/Alan-Robertson/qecode.