Quantum Circuit Optimisation Through Stabiliser Reduction of Pauli Exponentials

Master's Thesis

Candidate Number: 1060950

Your Degree: MSc Advanced Computer Science

Date: Trinity Term 2022

Word Count: 19,581 (via TeXcount)

Abstract

Quantum circuits require careful and in-depth optimisation to be run on existing quantum computers. This project investigates the application of the Product Rotation Lemma, introduced by Will Simmons in [32], as a tool for general circuit optimisation. These optimisations exploit the initial state information to identify redundancies in the circuit. This project delves into the lemma's applications for the Clifford-Pauli-Exponential form and its synthesis alongside the practical applications of this theory. The main contributions from this project are (1) the Pauli DAG Merging Theorem and its surrounding theory that gives simple necessary, and sufficient conditions for applying the Product Rotation Lemma in a strictly beneficial way and (2) providing results strongly indicating the feasibility of the techniques presented here as tools to be used in generic compilation procedures. This report provides proof of the novel theory coming out of this project, alongside a design and implementation of an end-to-end QASM-to-QASM compiler using staq as the base [3]. This compiler is very efficient on large circuits and provides CX count, and depth reductions of up to 42% after existing methods are applied.

Acknowledgements

This project was only made possible by the exceptional dedication and support provided by Will Simmons throughout the project; you have been a fantastic supervisor. You have gone above and beyond assisting with the project, your expertise in the area proving invaluable on numerous occasions. You gave so much of your knowledge, so many papers, and so much of your time to help me; I am extremely grateful for that.

Thanks also to Aleks Kissinger for his support in connecting Will and I and harbouring so many MSc students under his wing.

Thanks also to John van de Wetering and Lia Yeh for helping set up and continuing weekly meetings amongst the MSc students to discuss our projects and support one another. These meetings were incredibly beneficial.

Finally, thanks go to my friends and family for continually understanding that sometimes "I have to do some dissertation work" that has persisted for the past couple of months. They always support me, and for that, I am eternally grateful.

Contents

Chapter 1

Introduction

Quantum computing is a topic that has been explored extensively in recent years to create a tool to allow us to compute answers not efficiently possible through classical computation. The quantum circuit model for quantum computation is the most well-known and widely used model and acts as a direct analogue to traditional circuits. However, instead of classical bits of 0 or 1, we have qubits that can represent superpositions of such values. Instead of classical gates such as $\vee$ and $\wedge$, quantum gates such as rotations and controlled-NOT (CNOT) gates are used. A subset of these gates, namely the Clifford set, is known to be efficiently classically simulable from the Gottesman-Knill theorem [14], a crucial fact for some optimisation procedures.

After creating a circuit, a quantum computer can efficiently apply the gates and measurements to some initial quantum unit called a state. The results of the computation are the outcomes of the measurements performed. One method for creating a quantum computer is via superconductivity appearing at low temperatures. However, superconducting qubits have a short lifetime, and quantum gates have a chance to produce the wrong result. These issues are solvable

through Fault-Tolerant Quantum Computing but require far more qubits than available [18, 31, 34]. These conditions place us in the Noisy Intermediate-Scale Quantum (NISQ) era, where we cannot avoid these errors or "noise" during our computation. Having to permit noise is where quantum compilation comes into play, transforming some higher-level abstraction of a circuit to an optimised set of gates for a specific quantum computer to apply. The compiler's optimisations aim to reduce both the gate count and the depth of the circuit as computable proxies for reducing noise during computation.

One of the most common applications of quantum computing in the NISQ era is to compute the ground state energy of a Hamiltonian [27]. This computation is a fundamental problem in quantum chemistry and condensed matter physics. Quantum computers solve this problem by the Trotterization of the Hamiltonian and application of a Variational Quantum Eigensolver (VQE) [15]. For this context, we work with circuits consisting of Pauli exponentials and some Clifford component, a natural representation for circuits obtained from Trotterized Hamiltonians. Current compilation strategies for these circuits take the initial set of Pauli exponentials, use stabilisers to taper off qubits, choose some ordering for the Pauli exponentials, and finally synthesise the Pauli exponentials into a circuit [8, 25]. Commonly, this synthesis uses techniques like diagonalisation and phase polynomial synthesis, discussed in further detail throughout the project.

Outside of the context of VQE circuits, stabiliser reduction has not seen much use as a technique. However, it turns out that we can efficiently convert any quantum circuit into the same Pauli Exponential and Clifford component form [37]. In this form, we cannot freely change Pauli exponential's order without changing the circuit's meaning, so we must adapt this method to work in our new context. Introduced in [32] by Will Simmons, the Product Rotation Lemma provides the rules we need. This lemma assumes knowledge of the initial state and uses

stabilisers of this state to alter the Pauli exponentials. This lemma has not yet been studied for usage as a tool for general optimisation, so we seek to answer this question by testing the limits of this technique by applying it exhaustively to a given circuit. By choosing the exhaustive approach to this problem, we can provide insights into whether this technique is worth pursuing and offer tooling to assist with future work.

1.1 Summary of the Project

This project investigates the power of the Product Rotation Lemma as a general-purpose feature in current compilation toolchains. To do this, the theory around applying the Product Rotation Lemma to circuits split into Pauli exponentials with a Clifford component was first thoroughly investigated. This investigation led to a new rule dubbed the Pauli DAG Merging Theorem, providing an efficient check for applying the Product Rotation Lemma to combine two Pauli exponentials. Under a set of assumptions, we further show that the Pauli DAG Merging Theorem is a sound and complete rule for merging using the Product Rotation Lemma. The theory around applying the Product Rotation Lemma to reduce the complexity of the resulting circuits was also explored, with promising results applicable to heuristics.

These findings allowed the creation of an efficient end-to-end compiler using an existing toolkit in staq as a base [3]. This compiler can exhaustively perform merges using the Pauli DAG Merging Theorem and uses heuristics to apply the Product Rotation Lemma to improve the final circuit. We further show how this method applies to Clifford circuits, a technique used by the compiler to improve the final circuit. Alongside new techniques, the compiler also includes a series

of existing procedures to complete the compilation pipeline and offer a realistic look into how useful this tool is in practice. We benchmark this compiler and show that the techniques give significant improvements for specific classes of circuits. The compiler also runs within very reasonable time frames, allowing for use in real-world systems.

1.2 Structure of the Report

This report details the project's results, beginning with preliminary notation and background in Chapter 2. In this section, we detail existing knowledge, with no novel work being presented. Then, Chapter 3 builds on the background to provide a plethora of new theoretical results, including the Pauli DAG Merging Theorem. Next, the design and implementation of the full compiler are given in Chapter 4 and Chapter 5 respectively. Comprehensive benchmarking data is then presented and analysed in Chapter 6. Finally, Chapter 7 wraps up the report and provides avenues for future work to extend this project.

Chapter 2

Background

Before the main content of this report, the following chapter contains some background for the project to frame it in the current research. As part of this, basic definitions, preliminary proofs, and examples will be given to assist with the understanding of the later chapters. First, we shall introduce quantum computation in more detail; next, we give context on Pauli exponentials, Cliffords, and the Clifford-Pauli-Exponential form; before finally looking in more depth at current research in this area and the impact this work could have.

2.1 Quantum Computation

Quantum computation is a new and expanding technique to take advantage of quantum effects such as entanglement to achieve more efficient computation than is currently possible by regular, classical computers.

2.1.1 Classical and Quantum Bits

Quantum bits, or qubits, are the primary resource that makes quantum computation possible. Compared to the bits we find in computers, referred to as classical bits or just bits, qubits do not restrict themselves to existing in strictly *either* 0 or 1, but instead can be placed into what is called a *superposition* of the two. They are neither 0 nor 1 but rather exist simultaneously as a coherent combination of both. More concretely, in quantum computation, we represent 0 and 1 as the quantum basis states $|0\rangle$ and $|1\rangle$. A quantum state represents the current state that the system exists in. For pure quantum states, we can write them as a linear combination of our two basis states.

Definition 2.1 (Pure Quantum State)**.** *A pure quantum state* $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ *where* $\alpha, \beta \in \mathbb{C}$ *and* $|\alpha|^2 + |\beta|^2 = 1$.

Alongside pure states, we also have mixed quantum states which are probabilistic mixtures of any other quantum state. In other words, a mixed state has a probability distribution over pure states. The Bloch sphere represents all the states in which a single qubit can be; see Figure 2.1. The points on the surface of the sphere are the pure states, and those inside are mixed states. As seen on the sphere, there are three primary axes about which the sphere is rotated, X, Y, and Z. The sphere's north pole is usually allocated to $|0\rangle$, while the south pole of the sphere is allocated to $|1\rangle$.

To gain information about a quantum state, we perform a measurement with respect to the Z-basis. When measuring a quantum state, we non-deterministically measure one of the $|0\rangle$ or $|1\rangle$ states. The probability with which we observe $|0\rangle$ or $|1\rangle$ as the measurement outcome is defined as the proximity to each state on the Bloch sphere. As a result, any quantum state appearing on the equatorial

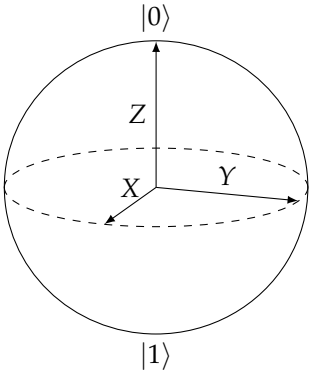


Figure 2.1: The Bloch sphere whose surface represents pure single-qubit quantum states

slice will cause $|0\rangle$ and $|1\rangle$ to be observed with a probability 0.5. More precisely, the probabilities obey the Born rule [23]. We do not use measurements or mixed quantum states for the remainder of this report, so no further context is given. When referring to a quantum state, it will be assumed to be a pure state.

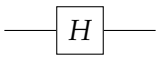
A single qubit alone is not very interesting, so we can also consider a combination of qubits that exist in parallel. When writing the states for these qubits, we can write them as $|0\rangle \otimes |0\rangle$. However, this notation is quite heavy, so for basis states, we often abbreviate this to $|0, 0\rangle$ or just $|00\rangle$.

2.1.2 Quantum Gates

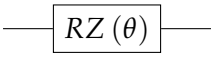
Once we have a quantum state, we need a way to transform that state. To do this, we introduce quantum gates. These gates can be applied to a single qubit or multiple qubits to evolve the represented state. The three main quantum gates are the Z-Rotation $RZ(\theta)$, Hadamard H, and Controlled-NOT CX gates.

The H gate is a single qubit gate that converts Z-basis states to X-basis states and vice versa. This gate effectively swaps the Bloch sphere's X and Z axes. This swap means that applying an H gate just before an $RZ(\theta)$ gate followed

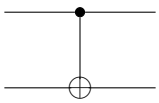
by a H gate causes the application of the rotation to the X axis. This behaviour results from the Z axis being the X axis during the rotation since it is performed *after* the swap. Since the H gate is self-inverse, performing this gate sequence is equivalent to basis transformation by conjugation.



The $RZ(\theta)$ gate is a single qubit gate that rotates the Z axis of the Bloch sphere by θ radians. Note that this leaves $|0\rangle$ and $|1\rangle$ where they are but will cause a rotation for every other quantum state.



The CX gate applies to two qubits; one is the target, and the other is the control. Assuming both qubits are Z-basis states, a CX gate results in a NOT gate being applied to the target qubit if the control qubit is in the $|1\rangle$ state. This behaviour is not very interesting over basis states, but over more complex quantum states, this gate enables the phenomenon known as *entanglement*. Entanglement occurs when two qubits are part of a single quantum state such that the state cannot be written as the parallel composition of individual single qubit states.



Like states, these gates can similarly be composed in parallel using $\otimes$. We can also specify which qubits of the state to which we wish to apply the gate, equivalent to parallel composing with the identity gate. For single qubit gates, we set the qubit as H_i to apply to the i^{th} qubit. For multiple qubit gates like the

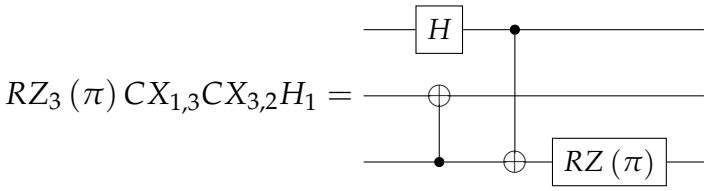


Figure 2.2: An example quantum circuit

CX gate, we write $CX_{i,j}$ to apply the control to qubit i and the target to qubit j. Indexing on gates will always occur from 1. As shown in Figure 2.2, we write the composition of gates such that the first gate applied appears at the end of the sequence, a notation commonly used for quantum circuits. The set of gates $RZ(\theta)$, Hadamard H, and Controlled-NOT CX are sufficient to describe any quantum computation [23]. I.e., they are a universal gate set for quantum computation. As a result, these will be the main gates considered for the remainder of this report.

2.1.3 Quantum Circuits as Linear Maps

Conveniently, we can model qubit-based quantum circuits as linear maps over $\mathbb{C}^2$. We begin by defining the Z-basis states $|0\rangle$ and $|1\rangle$ where a natural choice is as follows.

Definition 2.2 (Z-Basis Vectors)**.**

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad\qquad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

We can also define the X-basis states as a combination of $|0\rangle$ and $|1\rangle$. By convention, these are denoted $|+\rangle$ and $|-\rangle$.

Definition 2.3 (X-Basis Vectors)**.**

$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix} \qquad |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\-1\end{bmatrix}$$

We can now define the Hadamard gate (the H gate). Recalling that it transforms between the Z- and X- bases, it is clear that the definition should be the following.

Definition 2.4 (H Gate)**.**

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}$$

We also define the Z-Rotation gate (the $RZ(\theta)$ gate) in matrix form as follows.

Definition 2.5 ($RZ(\theta)$ Gate)**.**

$$RZ(\theta) = \begin{bmatrix}e^{-i\frac{\theta}{2}} & 0\\0 & e^{i\frac{\theta}{2}}\end{bmatrix}$$

Finally, we can define the CX gate over two qubits. For simplicity, we will assume the CX gate is applied to adjacent qubits of the state. This assumption is not required in practice.

Definition 2.6 (CX Gate)**.**

$$CX = \begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 0 & 1\\0 & 0 & 1 & 0\end{bmatrix}$$

Finally, we must define our gates' and states' sequential and parallel composition within linear maps. We can apply a quantum gate to a state by multiplying

the state by the gate through regular matrix multiplication. This matrix multiplication is referred to as sequential composition. As matrix multiplication occurs left to right, we write circuits in this reverse order, as noted previously. E.g., $H|0\rangle$ rather than $|0\rangle H$. We also need to consider the parallel composition of states and gates. To do this, we can use the $\otimes$-product, or tensor product, over linear maps.

Before moving on, we define the $RX(\theta)$, $RY(\theta)$ gates and some other commonly used gates. The $RX(\theta)$ and $RY(\theta)$ gates perform analogously to the $RZ(\theta)$ gate, performing a rotation of θ radians around the X and Y axes respectively.

Definition 2.7 ($RX(\theta)$ Gate)**.**

$$RX(\theta) = \begin{bmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}$$

Definition 2.8 ($RY(\theta)$ Gate)**.**

$$RY(\theta) = \begin{bmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}$$

Some other commonly used gates are as follows.

Definition 2.9 (Common Gates)**.**

$$\begin{aligned} S &\approx RZ\left(\frac{\pi}{2}\right) & S^\dagger &\approx RZ\left(-\frac{\pi}{2}\right) \\ SX &\approx RX\left(\frac{\pi}{2}\right) & SX^\dagger &\approx RX\left(-\frac{\pi}{2}\right) \\ T &\approx RZ\left(\frac{\pi}{4}\right) & T^\dagger &\approx RZ\left(-\frac{\pi}{4}\right) \end{aligned}$$

In this context, † refers to the conjugate transpose for linear maps over $\mathbb{C}^2$. A helpful property of quantum gates is that they are unitary matrices. In other

words, we have that $U^\dagger U = UU^\dagger = I$ for any unitary U and, by extension, quantum gate. We use $\approx$ to denote equality up to a global phase. Since all outputs from a quantum computation come from measurements obeying the Born rule, all global phases are not observable so they can be ignored [23].

2.2 Pauli Notation

An essential notion for this work is Pauli strings. These are tensor products of what is referred to as the Pauli matrices: I, Z, X, and Y.

Definition 2.10. *The Pauli matrices.*

$$I := \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad Z := \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad X := \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad Y := \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

The Pauli matrices are all members of the Pauli group P_1 acting on a single qubit. Specifically, they act as generators for the entire group under matrix multiplication. We have that the Pauli matrices satisfy the following lemmas.

Lemma 1: *The Pauli matrices are self-inverse. I.e.,*

$$II = I \qquad ZZ = I \qquad XX = I \qquad YY = I$$

Lemma 2: $IP = PI = P$, *for* $P \in \{Z, X, Y\}$. *I.e.,*

$$IZ = Z \qquad ZI = Z$$
$$IX = X \qquad XI = X$$
$$IY = Y \qquad YI = Y$$

Lemma 3: *Multiplying the non-identity Pauli matrices satisfy the following equations.*

$$ZX = iY \qquad XZ = -iY$$
$$XY = iZ \qquad YX = -iZ$$
$$YZ = iX \qquad ZY = -iX$$

Relating the Pauli matrices to our original gate-set, we have the following lemma.

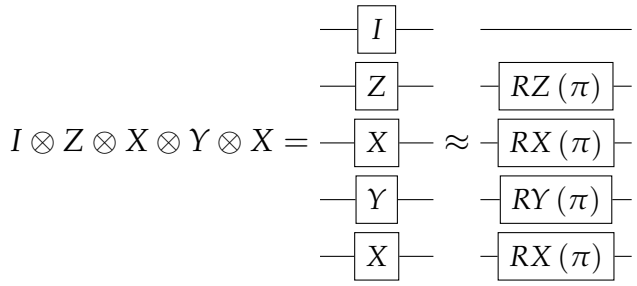
Lemma 2.1 (Pauli Matrices as Quantum Gates)**.**

$$Z \approx RZ(\pi) \qquad X \approx RX(\pi) \qquad Y \approx RY(\pi)$$
$$I = RZ(0) \qquad I = RX(0) \qquad I = RY(0)$$

These identities are easy to check numerically, so their proofs are omitted.

We combine these Pauli letters via the tensor product to produce full Pauli strings forming the group P_n, often omitting the tensor product in the notation in favour of vector notation for brevity. We choose a vector notation here to differentiate between the sequential and parallel composition of Pauli matrices. For example, the Pauli string $\overrightarrow{IZXYX}$ is given below. In cases where the Pauli string is sparse, we use the same subscript notation as for gates to omit identity letters of the string. Often, Pauli strings may need to be negated, i.e., $-\overrightarrow{IZXYX}$. In many cases, this negation is irrelevant or cumbersome to track, so we often

specify Pauli strings up to a negation.



We can also take the exponential of a given Pauli string $\vec{P}$, written $e^{i\theta\vec{P}}$ where θ is a phase. For notational purposes, Pauli exponentials may be denoted by $\vec{P}(\alpha) = e^{i\theta\vec{P}}$ where $\alpha = -2\theta$. A standard result of Pauli strings is the following lemma.

Lemma 4: *Pauli Exponential Expansion*

For a Pauli string $\vec{P}$, the following equation holds.

$$e^{i\theta\vec{P}} = \cos\theta I + i\sin\theta\vec{P}$$

To convert this into a circuit, we first need to convert all letters in the Pauli string to be Z or I. A Pauli exponential with a Pauli string of this form is referred to as a diagonal Pauli exponential. We can diagonalise an arbitrary Pauli exponential using the following lemma.

Lemma 5: *Given a Pauli exponential $e^{i\theta\vec{P}}$. We can conjugate the qubits containing X with H gates and Y with SX gates to obtain a diagonal Pauli exponential $e^{i\theta\vec{Q}}$.*

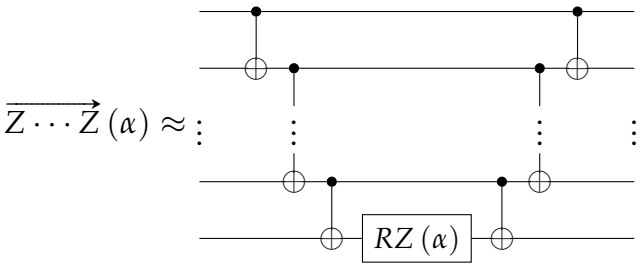
Proof of Lemma: We proceed inductively over the number of qubits of $\vec{P}$. The empty string case is trivial, so consider the first Pauli $\vec{P} = R \otimes \vec{P'}$ and let C be a Clifford unitary that diagonalises $\vec{P'}$ by induction. I.e., $C^\dagger\vec{P'}C = \vec{Q'}$ where $\vec{Q'}$ is diagonal. Let C_1 be a single qubit Clifford unitary such that $C_1^\dagger R C_1 \in \{I, Z\}$

then,

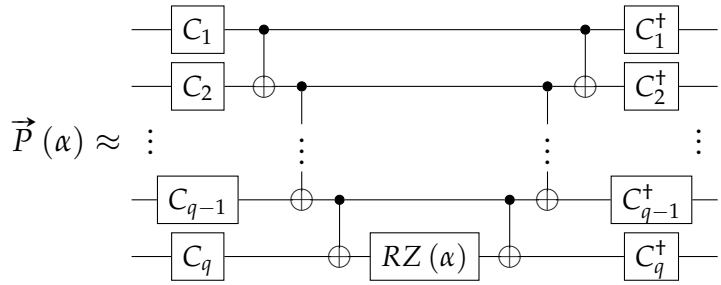
$$\begin{aligned}
\left(C_1^\dagger \otimes C^\dagger\right) e^{i\theta \vec{P}} \left(C_1 \otimes C\right) &= \left(C_1^\dagger \otimes C^\dagger\right) e^{i\theta\left(R \otimes \vec{P'}\right)} \left(C_1 \otimes C\right) \\
&= \left(C_1^\dagger \otimes C^\dagger\right) \left(\cos\theta I + i \sin\theta \left(R \otimes \vec{P'}\right)\right) \left(C_1 \otimes C\right) \\
&= \cos\theta I + i \sin\theta \left(C_1^\dagger \otimes C^\dagger\right) \left(R \otimes \vec{P'}\right) \left(C_1 \otimes C\right) \\
&= \cos\theta I + i \sin\theta \left(C_1^\dagger R C_1 \otimes C^\dagger \vec{P'} C\right) \\
&= \cos\theta I + i \sin\theta \left(C_1^\dagger R C_1 \otimes \vec{Q'}\right)
\end{aligned}$$

Since $\vec{Q'}$ is diagonal and $C_1^\dagger R C_1 \in \{I, Z\}$, $C_1^\dagger R C_1 \otimes \vec{Q'}$ is diagonal. Therefore, we conclude that any Pauli exponential can be diagonalised with an appropriate choice of C_1. Finally, it is easy to verify numerically that $SX^\dagger (Y) SX = Z$, $H(X) H = Z$, $I(Z) I = Z$, and $I(I) I = I$. ■

After converting the Pauli string to only contain Z and I, we are left with a phase polynomial (a sequence of diagonal Pauli exponentials). There is extensive literature surrounding phase polynomials providing the following synthesis of a single diagonal Pauli exponential into a circuit [4, 26, 33].



This combination allows us to synthesise an arbitrary Pauli exponential where C_i is defined as in the proof of Lemma 5.



Pauli strings and Pauli exponentials can also commute with one another. These rules allow for the reordering of Pauli exponentials within a circuit without changing the circuit's meaning.

Lemma 6: *Pauli Letter Commutation*

Let $P, Q \in \{Z, X, Y\}$. We have $PQ = QP$ iff $P = Q$ and $PQ = -QP$ otherwise.

Proof of Lemma: This can be seen via inspection of Lemma 2 and Lemma 3. ■

Lemma 7: *Pauli String Commutation*

Let $\vec{P}$ and $\vec{Q}$ be Pauli strings. $\vec{P}\vec{Q} = \vec{Q}\vec{P}$ iff the number of terms in $\vec{P}$ that anti-commute with their associated term in $\vec{Q}$ is even. Otherwise, $\vec{P}\vec{Q} = -\vec{Q}\vec{P}$.

Proof of Lemma: Each letter in the Pauli string either commutes or anti-commutes with the relative Pauli letter in the other Pauli string. For each case of anti-commutation, a -1 factor is produced by Lemma 6. Combining these gives us $\vec{P}\vec{Q} = (-1)^k \vec{Q}\vec{P}$ where k is the number of anti-commuting pairs. $(-1)^k = 1$ iff k is even. ■

For Pauli exponential commutation rules, we recall some elementary facts of matrix exponentiation. If $XY = YX$ then, $e^X e^Y = e^{X+Y}$. If Y is invertible then $e^{YXY^{-1}} = Ye^X Y^{-1}$.

Lemma 8: *Pauli Exponential Commutation*

Let $\vec{P}$ *and* $\vec{Q}$ *be Pauli strings, and let* α *and* β *be phases.* $\vec{P}(\alpha)\vec{Q}(\beta) = \vec{Q}(\beta)\vec{P}(\alpha)$ *if* $\vec{P}\vec{Q} = \vec{Q}\vec{P}$ *and* $\vec{P}(\alpha)\vec{Q} = \vec{Q}\vec{P}(\alpha)$ *if* $\vec{P}\vec{Q} = \vec{Q}\vec{P}$.

Proof of Lemma: Since $\vec{P}\vec{Q} = \vec{Q}\vec{P}$ we get the following.

$$\begin{aligned}
\vec{P}(\alpha)\vec{Q}(\beta) &= e^{-i\frac{\alpha}{2}\vec{P}}e^{-i\frac{\beta}{2}\vec{Q}} \\
&= e^{\left(-i\frac{\alpha}{2}\vec{P}\right)+\left(-i\frac{\beta}{2}\vec{Q}\right)} \\
&= e^{\left(-i\frac{\beta}{2}\vec{Q}\right)+\left(-i\frac{\alpha}{2}\vec{P}\right)} \\
&= e^{-i\frac{\beta}{2}\vec{Q}}e^{-i\frac{\alpha}{2}\vec{P}} \\
&= \vec{Q}(\beta)\vec{P}(\alpha)
\end{aligned}$$

We also get.

$$\begin{aligned}
\vec{P}(\alpha)\vec{Q} &= e^{-i\frac{\alpha}{2}\vec{P}}\vec{Q} \\
&= \vec{Q}\vec{Q}e^{-i\frac{\alpha}{2}\vec{P}}\vec{Q} \\
&= \vec{Q}e^{-i\frac{\alpha}{2}\vec{Q}\vec{P}\vec{Q}} \\
&= \vec{Q}e^{-i\frac{\alpha}{2}\vec{Q}\vec{Q}\vec{P}} \\
&= \vec{Q}e^{-i\frac{\alpha}{2}\vec{P}} \\
&= \vec{Q}\vec{P}(\alpha)
\end{aligned}$$

■

Lemma 9: *Pauli Exponential Fusion*

Let $\vec{P}$ *be a Pauli string and let* α *and* β *be phases.* $\vec{P}(\alpha)\vec{P}(\beta) = \vec{P}(\alpha+\beta)$.

Proof of Lemma: This follows directly from $e^X e^Y = e^{X+Y}$ for commuting X

and Y.

$$\begin{aligned}\vec{P}(\alpha)\vec{P}(\beta) &= e^{-i\frac{\alpha}{2}\vec{P}}e^{-i\frac{\beta}{2}\vec{P}} \\ &= e^{-i\frac{\alpha}{2}\vec{P}-i\frac{\beta}{2}\vec{P}} \\ &= e^{-i\left(\frac{\alpha+\beta}{2}\right)\vec{P}} \\ &= \vec{P}(\alpha+\beta)\end{aligned}$$

■

2.3 Clifford Gates and Stabilisers

An important subset of the universal gate set is the Clifford group. This set is formally defined as the unitary group that normalises the Pauli group.

Definition 2.11 (The Clifford Group)**.** *The Clifford group $\mathfrak{C}_n$ is defined as follows.*

$$\mathfrak{C}_n = \left\{C | C \cdot P_n \cdot C^\dagger = P_n\right\}$$

In other words, to check if a gate is in the Clifford group, we need to verify that conjugating any Pauli string with it corresponds to another Pauli string. The following generating gates satisfy this property.

Lemma 10: *The S, CX, and H gates act as a generating set for all Clifford gates.*

This standard result in quantum computing comes from the Gottesman-Knill theorem [14] and allows us to define a Clifford state.

Definition 2.12 (Clifford State)**.** *A state $|\psi\rangle$ is a Clifford state if it can be written as the composition of $|0\rangle^{\otimes q}$ and a series of S, CX, and H gates.*

Another important concept is that of a stabiliser. We define stabilisers to act over states as follows.

Definition 2.13 (Stabilisers)**.** *Given a state $|\psi\rangle$, a unitary U is a stabiliser of $|\psi\rangle$ if $U|\psi\rangle = |\psi\rangle$.*

A useful property of stabilisers is that two stabilisers of the same state $|\psi\rangle$ always commute.

Lemma 11: *Given a state $|\psi\rangle$ and unitaries A and B that are stabilisers of $|\psi\rangle$, $AB = BA$.*

Proof of Lemma: Assume for the sake of contradiction that $AB = -BA$.

$$\begin{aligned} AB = -BA &\implies AB|\psi\rangle = -BA|\psi\rangle \\ &\implies A|\psi\rangle = -B|\psi\rangle \\ &\implies |\psi\rangle = -|\psi\rangle \end{aligned}$$

This is a contradiction so $AB = BA$. ■

Stabilisers allow us to add or remove gates from a state without affecting its meaning. Starting from a state of $|0\rangle^{\otimes q}$, it is easy to verify that Z_i is a stabiliser of this state. Since this is our starting state, we can make the following claim about Clifford states.

Lemma 12: *Pauli strings stabilise Clifford states.*

Proof of Lemma: It is easy to verify that $|\psi\rangle = |0\rangle^{\otimes q}$ is stabilised by Pauli strings containing only Z and I. We now have the following for any Clifford unitary C, Pauli string $\vec{P}$ such that $\vec{P}|\psi\rangle = |\psi\rangle$, and Pauli string $\vec{Q}$ such that

$\vec{Q} = C\vec{P}C^\dagger$.

$$\begin{aligned}\vec{Q}C|\psi\rangle &= C\vec{P}C^\dagger C|\psi\rangle \\ &= C\vec{P}|\psi\rangle \\ &= C|\psi\rangle\end{aligned}$$

■

Despite many non-Pauli strings stabilising Clifford states, we restrict stabilisers to Pauli strings for the remainder of this report.

2.4 Clifford-Pauli-Exponential Form

This project aims to investigate the power of the Product Rotation Lemma in the context of state optimisation. The full Product Rotation Lemma is as follows.

Lemma 13 (Product Rotation Lemma): *Let A and B be commuting operators such that $B\mathscr{C} = \mathscr{C}$ for some linear map $\mathscr{C}$. Then $e^{i\theta A}\mathscr{C} = e^{i\theta AB}\mathscr{C}$.*

The proof of this lemma can be found in [32], but for this context, we only need to consider a simplification of this lemma.

Lemma 14 (Pauli Product Rotation Lemma): *Let $\vec{P}$ and $\vec{Q}$ be commuting Pauli strings such that $\vec{Q}|\psi\rangle = |\psi\rangle$ for some state $|\psi\rangle$. Letting $\vec{R} = \pm\vec{P}\vec{Q}$, then $\vec{P}(\alpha)|\psi\rangle = \vec{R}(\pm\alpha)|\psi\rangle$*

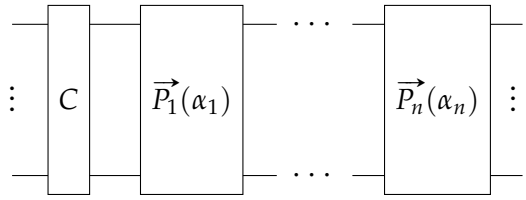
Proof of Lemma: Using the expansion of Pauli exponentials, we get the following.

$$\begin{aligned}
\vec{P}(\alpha)|\psi\rangle &= e^{-i\frac{\theta}{2}\vec{P}}|\psi\rangle \\
&= \left(\cos\frac{\theta}{2}I + i\sin\frac{\theta}{2}\vec{P}\right)|\psi\rangle \\
&= \cos\frac{\theta}{2}|\psi\rangle + i\sin\frac{\theta}{2}\vec{P}|\psi\rangle \\
&= \cos\frac{\theta}{2}|\psi\rangle + i\sin\frac{\theta}{2}\vec{P}\vec{Q}|\psi\rangle \\
&= \left(\cos\pm\frac{\theta}{2}I + i\sin\pm\frac{\theta}{2}\vec{R}\right)|\psi\rangle \\
&= e^{\mp i\frac{\theta}{2}\vec{R}}|\psi\rangle \\
&= \vec{R}(\pm\alpha)|\psi\rangle
\end{aligned}$$

We require $\vec{P}$ and $\vec{Q}$ to commute as Lemma 4 requires the exponentiated operator to be self-inverse. ■

This application of the Product Rotation Lemma requires the stabiliser to be a Pauli string. We have just seen that this is a property held by Clifford states. As a result, it is natural to consider circuits containing a Clifford state and a series of Pauli exponentials that we can manipulate using the Product Rotation Lemma. We refer to this form as the Clifford-Pauli-Exponential form of a state. We can omit the state without loss of generality and assume it to be $|0\rangle^{\otimes q}$ to generalise the definition to a circuit form defined as follows.

Definition 2.14 (Clifford-Pauli-Exponential Form)**.** *A quantum circuit is in Clifford-Pauli-Exponential form if it consists of a series of Clifford gates, followed by a series of Pauli exponentials. More formally, there exists a Clifford unitary C and Pauli exponentials* $\vec{P_1}(\alpha_1), \ldots, \vec{P_k}(\alpha_k)$ *such that the circuit is* $P_1(\alpha_1)\cdots\vec{P_k}(\alpha_k)\,C$*. Or equivalently,*



Initially, most circuits are not of this form, so we need some conversion method. For simplicity, we assume that the gates in the given circuit are either Z-Rotations (i.e., $RZ(\pm\theta)$ gates), Hadamards (i.e., H gates), or CNOTs (i.e., CX gates). This gate set is known to be universal, so this is done without loss of generality [23]. Considering each gate, the only non-Clifford gates are $RZ(\theta)$ where θ is not an integer multiple of $\frac{\pi}{2}$. Since they are non-Clifford, they need to be converted to a Pauli exponential.

Lemma 15: $RZ_i(\theta) \approx \overrightarrow{Z_i}(\theta)$.

Proof of Lemma: This is true by definition. ■

Using Lemma 15 we can convert our circuit into a sequence of Pauli exponentials and Clifford terms. It only remains to show how all Clifford terms can be pulled to the start of the circuit.

Lemma 16: *Consider a Clifford unitary C, and Pauli exponential $\vec{P}(\alpha)$. Let $\pm\vec{Q} = C\vec{P}C^\dagger$. Then $C\vec{P}(\alpha) = \vec{Q}(\pm\alpha)C$.*

Proof of Lemma: This follows directly from $e^{YXY^{-1}} = Ye^XY^{-1}$ for invertible Y.

$$\begin{aligned} C\vec{P}(\alpha) &= C\vec{P}(\alpha)C^\dagger C \\ &= Ce^{-i\frac{\alpha}{2}\vec{P}}C^\dagger C \\ &= e^{-i\frac{\alpha}{2}C\vec{P}C^\dagger} \\ &= e^{\mp i\frac{\alpha}{2}\vec{Q}}C \\ &= \vec{Q}(\pm\alpha)C \end{aligned}$$

■

Using Lemma 16, it only remains to show how to calculate $C\vec{P}C^\dagger$ for every Clifford C and Pauli string $\vec{P}$. We know this to be possible for every Pauli string by the definition of the Clifford group. Since CX, H, and S form a complete gate set for Cliffords [14], it suffices to only show the calculations for $CX\left(\vec{P}\right)CX$, $H\left(\vec{P}\right)H$, and $S\left(\vec{P}\right)S^\dagger$. We first introduce the following lemma to assist with calculations.

Lemma 17: *For any unitary U and Pauli string $\vec{P}$ such that $\vec{P} = \alpha\vec{P_1}\vec{P_2}$ and $U\vec{P_i}U^\dagger = \beta_i\vec{Q_i}$ for $i \in \{1,2\}$, then $U\vec{P}U^\dagger = \alpha\beta_1\beta_2\vec{Q_1}\vec{Q_2}$.*

Proof of Lemma:

$$U\vec{P}U^\dagger = \alpha U\vec{P_1}\vec{P_2}U^\dagger = \alpha U\vec{P_1}U^\dagger U\vec{P_2}U^\dagger = \alpha\beta_1\beta_2\vec{Q_1}\vec{Q_2}$$

■

Lemma 18: *The following S conjugations hold.*

$$SZS^\dagger = Z \qquad SXS^\dagger = Y \qquad SYS^\dagger = -X$$

Proof of Lemma: It is easy to verify $SZS^\dagger = Z$ and $SXS^\dagger = Y$ by calculation. By applying Lemma 17 we get $S^\dagger YS = -iZY = -X$. ■

Lemma 19: *The following H conjugations hold.*

$$HZH = X \qquad HXH = Z \qquad HYH = -Y$$

Proof of Lemma: It is easy to verify $HZH = X$ and $HXH = Z$ by calculation. By applying Lemma 17 we get $HYH = -Y$. ■

Lemma 20: *The following CX conjugations hold, assuming the first qubit is the control and the second qubit is the target.*

$$CX\,(IZ)\,CX = ZZ \qquad CX\,(IX)\,CX = IX \qquad CX\,(IY)\,CX = ZY$$
$$CX\,(ZI)\,CX = ZI \qquad CX\,(XI)\,CX = XX \qquad CX\,(YI)\,CX = YX$$
$$CX\,(ZZ)\,CX = IZ \qquad CX\,(XX)\,CX = XI \qquad CX\,(YY)\,CX = -XZ$$

$$CX\,(ZX)\,CX = ZX \qquad CX\,(ZY)\,CX = IY$$
$$CX\,(XY)\,CX = YZ \qquad CX\,(XZ)\,CX = -YY$$
$$CX\,(YZ)\,CX = XY \qquad CX\,(YX)\,CX = YI$$

Proof of Lemma: The equations for IZ, ZI, IX, and XI are easily verified numerically. The remaining equations follow by the application of Lemma 17. ■

Bringing this all together, we get the following theorem.

Theorem 1: *Any circuit of q qubits containing c Clifford gates and k non-Clifford $RZ\,(\theta)$ gates can be converted into an equivalent Clifford-Pauli-Exponential form in $\mathcal{O}\left(\min\left(c, q^2\right) \cdot k\right)$ conjugations of Pauli exponentials.*

Proof of Theorem: We can use the following procedure.

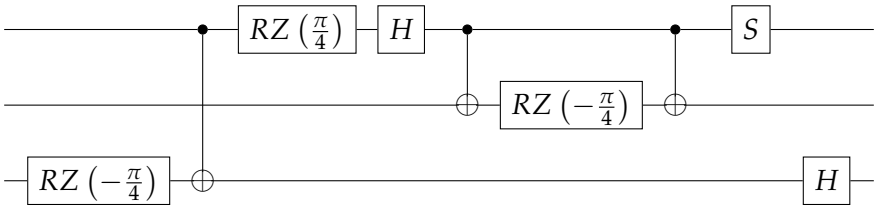
1. Convert non-Clifford $RZ_i\,(\theta)$ to $Z_i\,(\theta)$ Pauli exponentials using Lemma 15
2. Use Lemma 16, Lemma 18, Lemma 19, and Lemma 20 to bring all Cliffords to the start of the circuit

All Cliffords are now at the start of the circuit, and all other gates are Pauli exponentials; thus, we are in Clifford-Pauli-Exponential form. Each Pauli exponential

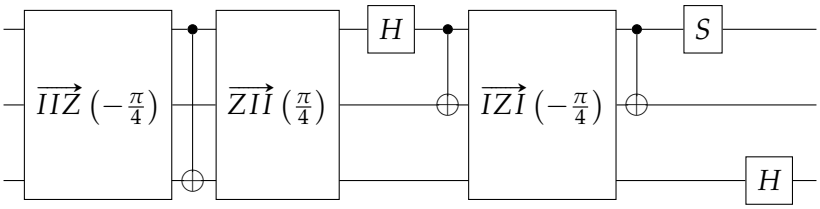
is conjugated by each Clifford at most once, giving $\mathcal{O}(c \cdot k)$ conjugations total.

We can improve on this when $l \cdot q^2 < c$ for some constant l by recognising that for every Clifford circuit, there is an equivalent circuit using at most $O(q^2)$ gates that can be found efficiently [1]. For each Pauli exponential, we can first compress all Cliffords that appear after it to $O(q^2)$ gates before pushing them through. This gives the desired bound of $\mathcal{O}(\min(c, q^2) \cdot k)$. ■

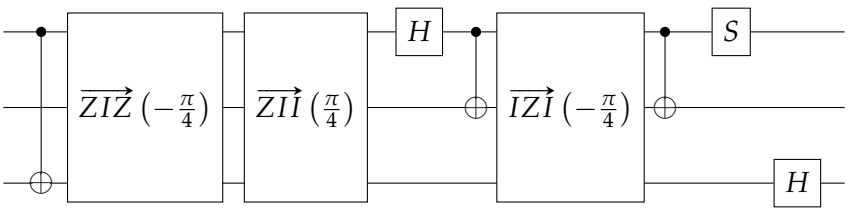
To illustrate this procedure, we consider the following circuit.



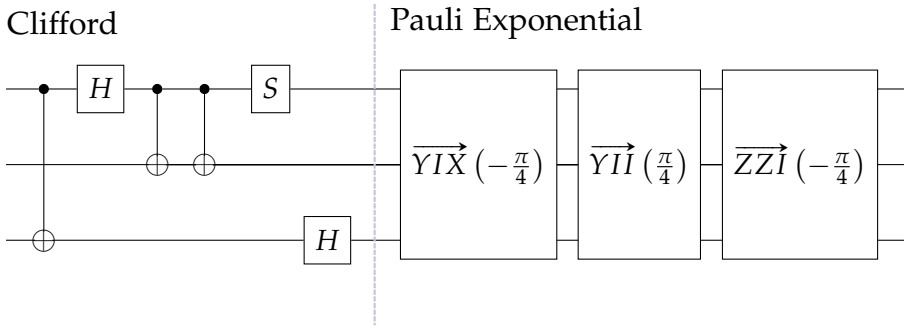
We apply the first step of the procedure to convert the circuit to only contain Cliffords and Pauli exponentials giving the following circuit.



We can apply Lemma 20 to the first Pauli exponential and first CX gate. Since the CX gate applies to the first and third qubit, we apply the $(CX)\, IZ\, (CX) = ZZ$ rule, resulting in the following circuit.



We can repeat this process, leading to the following final circuit in Clifford-Pauli-Exponential form.



2.5 Related Work

Before continuing to the main content of this report, we will discuss some related works and their relations to this technique.

2.5.1 Phase-Folding

Introduced by Matthew Amy in their PhD thesis, phase-folding is a technique for merging non-Clifford gates by computing their phase polynomial [2]. The notation for phase polynomials is very involved and will not be listed here. The algorithm proceeds in two steps, first performing phase analysis to compute the phase polynomial of the circuit. Then, the phase-folding step takes place, where all non-Clifford phases in the phase polynomial that are equivalent are combined into a single phase. A technique called phase teleportation is very similar to this. Introduced by Aleks Kissinger and John van de Wetering, phase teleportation uses the rules of the ZX-calculus to locate phases that are about to fuse using the rewrite rules, allowing them to be merged in the original circuit [16].

The two phase-folding and phase teleportation algorithms are equivalent when comparing only the non-Clifford count.

A generalisation of the phase-folding algorithm by Fang Zhang and Jianxin Chen from 2019 first converts the circuit to the Clifford-Pauli-Exponential form [37]. Then, using the Pauli exponential commutation rules of Lemma 8, the Pauli exponentials are rearranged to allow trivial merges by Lemma 9. This paper observes that the phase teleportation algorithm is equivalent to the new algorithm proposed in terms of non-Clifford count giving equivalence between all three algorithms. The main difference in the techniques is that the phase-folding and phase teleportation algorithms do not alter any structure of the original circuit, simply merging any permitted phases. In contrast, the Clifford-Pauli-Exponential form algorithm requires the re-synthesis of the Pauli exponentials into quantum gates to rebuild the circuit after the merging step. The importance of *not* having to re-synthesise the circuit is that it allows any circuit to be passed to the algorithm without affecting important metrics like gate depth and gate count.

2.5.2 Clifford-Pauli-Exponential Form Usage

Usage of the Clifford-Pauli-Exponential form is quite common within the literature. It commonly appears that the presence of Clifford gates within the circuit is irrelevant for metrics, most commonly when optimising the T-count for a circuit.

As mentioned, the paper by Fang Zhang and Jianxin Chen from 2019 uses a near identical form as seen in this work [37]. They first translate the initial circuit to Clifford-Pauli-Exponential form before translating the set of Pauli exponentials into a DAG form. They utilise this DAG form to reduce both the T-count, as

mentioned previously, and the T-depth of the resulting circuit. The T-depth is reduced by identifying that the minimum T-depth attainable in the circuit equals the length of the longest path in the DAG. During their construction phase, the paper details the same merging technique as given above by exploiting the commutativity rules of Pauli exponentials.

This form is also used in a paper by David Gosset et al. from 2013, where the optimal T-Count for circuits is found using an exponential search [13]. As part of this procedure, they prove that any unitary can be written in a variation of Clifford-Pauli-exponential form with tighter restrictions called the channel representation. This representation uses Pauli exponentials as a set of basis functions which are then searched over for an exact match to construct a T-optimal implementation of an arbitrary Clifford+T unitary.

We also see this form come up in the restricted context of measurement pattern circuit extraction [32]. This work by Will Simmons collects the Clifford gates at the beginning of the circuit giving access to the Pauli stabilisers of the circuit. This work introduced the Product Rotation Lemma as a method for modifying Pauli exponentials to move the rotation setting the Pauli basis of a measurement to the output of the measurement pattern[1]. This work provides an extraction algorithm using this technique and proves that it succeeds when the initial measurement pattern has a Pauli flow.

Another similar form to the Clifford-Pauli-Exponential form exists where the Clifford portion appears *after* the Pauli exponentials, which is also quite common. A paper by Daniel Litinski uses it as a convenient form for designing fault-tolerant for large-scale quantum computing [21]. Since Cliffords are considered essentially free in this context, it is convenient to shift all Clifford gates

[1]This was also the first mention in the literature of using the Product Rotation Lemma to allow for additional Pauli exponential merges

to one end of the circuit to focus on the non-Clifford components.

2.5.3 Clifford-Pauli-Exponential Form Synthesis

A fundamental problem for this form is synthesising groups of Pauli exponentials. A wide range of research has been done on this topic ranging from smaller scale systems where pairs of rotations are synthesised simultaneously [7] to larger groupings where techniques such as diagonalisation come into play [8, 36]. For larger groupings, we can find commuting groups of Pauli exponentials that are as large as possible. In these groups, all Pauli exponentials commute with each other but are not diagonal. This form is not well suited for synthesis since it requires the Clifford conjugation of every Pauli on every qubit containing an X or a Y as shown in Lemma 5. These single qubit gates often get in the way of optimising CX gate layers. To reduce the cost of conjugating individual Pauli exponentials and to improve CX optimisation, we search for a single Clifford unitary that changes all Pauli exponentials to contain only I and Z terms once we conjugate the entire group. This process is called diagonalisation, for which a few different methods exist. This resulting form is a phase polynomial, and we can then use a standard phase polynomial synthesis algorithm such as Gray-Synth to complete the extraction [4].

The Gray-Synth algorithm is a heuristic for synthesising phase polynomials introduced by Matthew Amy et al. in [4]. This algorithm takes inspiration from Gray codes where only a single bit is changed while iterating through all elements of $\{0,1\}^q$.

2.5.4 Chemistry Circuits and Ansatz

Finally, we frame similar applications of this technique that currently exist. A typical application of quantum computing is for the computation of the ground state of an electron configuration for a molecule. A quantum-classical procedure called the Variational Quantum Eigensolver (VQE) algorithm uses the expectation value of a cost function applied to some parameterised circuit to approximate the energy, which it then attempts to minimise. This parameterised circuit, referred to as an ansatz, prepares the quantum state of the molecule and allows the VQE algorithm to tweak the parameters to minimise the cost function. The evolution of the electron configurations can be expressed as Hamiltonians presented as sums over Pauli strings. One method of modelling this evolution as a circuit is through the Trotterization of the Hamiltonians giving an ansatz consisting of many Pauli exponentials.

One software to assist with this is the OpenFermion library [25]. This library allows the generation and optimisation of quantum algorithms for simulating quantum chemistry. As part of this optimisation, the library uses similar stabiliser techniques to perform qubit tapering, where qubits unimportant to the computation can be removed [5].

Chapter 3

Theory

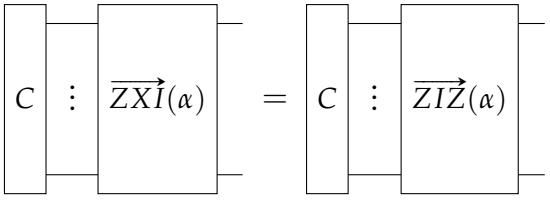
This chapter details a novel theory developed to explore applying the Product Rotation Lemma in varying contexts. The content of this chapter uses large amounts of the theoretical foundation set out in Chapter 2. A reader unfamiliar with any concepts is directed there for further explanation.

3.1 Product Rotation Lemma

Pauli strings being stabilisers of Clifford states as seen in Lemma 12 gives access to an instrumental technique for handling Pauli exponentials, the Product Rotation Lemma. The Product Rotation Lemma given in Lemma 14 allows us to augment the Pauli string of a Pauli exponential so long as we have a stabiliser that commutes with the string in the exponential.

For example, we can consider the following circuit of a Clifford state C composed with a Pauli exponential $\overrightarrow{ZXI}(\alpha)$. We define C such that $\overrightarrow{IXZ}$ is a stabiliser of C.

Then we get the following.



By applying the Product Rotation Lemma, we can reduce the number of Pauli exponentials in the circuit by merging them. We can perform this merging step by changing the string of one into the other. Then, the Pauli exponentials fuse, their phases adding together. This merging is formalised in the Stabiliser Merge Rule.

Lemma 21 (Stabiliser Merge Rule): *Let* $\vec{P}(\alpha)$ *and* $\vec{Q}(\beta)$ *be Pauli exponentials such that* $\vec{P}\vec{Q} = \vec{Q}\vec{P}$ *and* $|\psi\rangle$ *be a state such that* $\pm\vec{P}\vec{Q}|\psi\rangle = |\psi\rangle$*. Then,*

$$\vec{P}(\alpha)\,\vec{Q}(\beta)\,|\psi\rangle = \vec{P}(\alpha \pm \beta)\,|\psi\rangle$$

Proof of Lemma: Since $\vec{P}$ and $\vec{Q}$ are commuting, $\left(\vec{Q}\right)\left(\vec{P}\vec{Q}\right) = \left(\vec{P}\vec{Q}\right)\left(\vec{Q}\right)$. Therefore, we can apply Lemma 14 on $\vec{Q}(\beta)$ to transform $\vec{Q}(\beta)$ into $\vec{P}(\pm\beta)$.

$$\vec{P}(\alpha)\,\vec{Q}(\beta)\,|\psi\rangle = \vec{P}(\alpha)\,\vec{P}(\pm\beta)\,|\psi\rangle = \vec{P}(\alpha \pm \beta)\,|\psi\rangle$$

■

Finally, we can apply the Product Rotation Lemma to cancel out Pauli exponentials. This application changes the final circuit up to a global phase, but this is not observable in measurements, so it is permissible.

Lemma 22 (Stabiliser Cancel Rule): *Let* $\vec{P}(\alpha)$ *be a Pauli exponential and* $|\psi\rangle$ *be a state such that* $\pm\vec{P}|\psi\rangle = |\psi\rangle$*. Then* $\vec{P}(\alpha)\,|\psi\rangle \approx |\psi\rangle$*.*

Proof of Lemma: Since Pauli strings trivially commute with themselves, we can apply the Product Rotation Lemma on $\vec{P}(\alpha)$ to convert it to $\vec{I}$, where I is the identity Pauli.

$$\vec{P}(\alpha)|\psi\rangle = \vec{I}(\alpha)|\psi\rangle \approx |\psi\rangle$$

Where $\vec{I}$ is just the identity Pauli string on all qubits. ■

These applications of the Product Rotation Lemma are only applicable in the presence of a single Pauli exponential on a state, a very limited context. Therefore, we also wish to consider how this rule applies in a more general case. For the remainder of this section, it is essential to note the rule of commutation that is exclusively used.

Definition 3.1. *We say two Pauli exponentials $E_1 = \vec{P}(\alpha)$, $E_2 = \vec{Q}(\beta)$ commute iff $\vec{P}\vec{Q} = \vec{Q}\vec{P}$. This may also be written as $E_1E_2 = E_2E_1$.*

Without knowledge of a specific phase or Pauli string, this does hold. However, where the phase or the Pauli string is known up front, the *only if* portion does not hold. We can consider a Pauli exponential with phase 0, $\vec{P}(0)$. Since the phase is 0, this resolves to the identity and commutes with any Pauli exponential. For this context, we will use the above definition to govern commutation.

3.2 Pauli DAG

Given a circuit in Clifford-Pauli-Exponential form, the commutation laws between Pauli exponentials permit changes to the order of the Pauli exponentials without changing the semantics of the circuit. Therefore, we treat the sequence of Pauli

exponentials as a DAG to abstract the redundancy of total orderings. Before translation into a DAG, we have a sequence of Pauli exponentials. It is helpful to define such a sequence formally.

Definition 3.2 (Pauli Exponential Sequence)**.** *A Pauli exponential sequence is defined as a sequence* $\mathbb{E} = (E_1, ..., E_n)$ *where* $E_i = \overrightarrow{P_i}(\alpha_i)$ *together represent the circuit,*

$$\overrightarrow{P_n}(\alpha_n) \cdots \overrightarrow{P_1}(\alpha_1)$$

We denote the original circuit $C(\mathbb{E})$*. We say two sequences* $\mathbb{E}$ *and* $\mathbb{E}'$ *are equal under a state* $|\psi\rangle$ *if their circuits are equal up to a global phase when initialised by the state. I.e.,* $\mathbb{E} \approx_{|\psi\rangle} \mathbb{E}'$ *iff* $C(\mathbb{E})|\psi\rangle \approx C(\mathbb{E}')|\psi\rangle$*. We say two sequences* $\mathbb{E}$ *and* $\mathbb{E}'$ *are equal if their circuits are equal. I.e.,* $\mathbb{E} = \mathbb{E}'$ *iff* $C(\mathbb{E}) = C(\mathbb{E}')$*. We note that* $\mathbb{E} = \mathbb{E}'$ *implies* $\mathbb{E} \approx_{|\psi\rangle} \mathbb{E}'$ *for any state* $|\psi\rangle$*.*

With these sequences, it is helpful to define the construction of their associated DAG[1].

Definition 3.3 (Pauli DAG)**.** *Let* $\mathbb{E} = (E_1, ..., E_n)$ *be a Pauli exponential sequence. We define the Pauli DAG* $D(\mathbb{E}) = (V, A)$ *where* $V = \{v_1, ..., v_n\}$ *and*

$$A = \left\{ (v_i, v_j) \in V^2 | E_i E_j \neq E_j E_i \wedge i < j \right\}$$

Where it is unambiguous, D will be used in place of $D(\mathbb{E})$*.*

Note that this is *not* the transitive closure of the dependency graph between the exponentials but is also not minimal. Over the DAG, we also define three relations ρ, P and π representing different classes of parents.

[1]Interestingly, this is the same graph structure used in [37] for a different purpose

Definition 3.4 (Parent Sets). *Given a Pauli DAG $D = (V, A)$, we define $\rho_D : V \to 2^V$ as the set of vertices that are a direct parent D. More formally,*

$$\rho_D(v_i) = \{v_j \in V | (v_j, v_i) \in A\}$$

We can then define $\rho_D : 2^V \to 2^V$ over sets as the union of parents of all members of the set. More formally,

$$\rho_D(V') = \cup_{v_i \in V'} \rho_D(v_i)$$

Using this, we can also define $P_D^i : V \to 2^V$ as the set of parents at a distance at most i. More formally,

$$\begin{aligned} P_D^0(v_j) &= \rho(v_j), \\ P_D^i(v_j) &= \rho\left(P_D^{i-1}(v_j)\right) \cup P_D^{i-1}(v_j) \end{aligned}$$

Finally, we define $\pi_D : V \to 2^V$ as the set of Pauli exponentials that are a parent in the transitive closure of D. More formally,

$$\pi_D(v_i) = P_D^\infty(v_i)$$

Defining an ordering between vertices in a Pauli DAG is also helpful.

Definition 3.5 (Pauli DAG Ordering). *Given a Pauli DAG $D = (V, A)$, we define $\to_D$ over the set of vertices such $v_i \to_D v_j$ iff there is a path from v_i to v_j. More formally,*

$$v_i \to_D v_j \Leftrightarrow v_i \in \pi_D(v_j)$$

We can define $\sim_D$ over the set of vertices such that $v_i \sim_D v_j$ iff there does not exist a

path from v_i to v_j or vice versa. More formally,

$$v_i \sim_D v_j \Leftrightarrow v_i \not\rightarrow_D v_j \wedge v_j \not\rightarrow_D v_i$$

Where the choice of D is unambiguous, it will be omitted from the notation.

3.3 Pauli Sequence Rules

Now that we have a well-defined notion of a Pauli exponential sequence and Pauli DAGs, we can prove the following rules for modifying Pauli exponential sequences. This first lemma gives us an easy method for commuting Paulis throughout a Pauli exponential sequence $\mathbb{E}$ by permitting permutations that associate with a topological ordering in the DAG. For this lemma, we first need to describe the notation for permutation functions.

Definition 3.6 (Permutation Functions)**.** *A permutation function $\Omega : [1, n] \to [1, n]$ is a bijective function representing a permutation of $(1, \ldots, n)$. We denote the resulting position of element i in the permutation as $\Omega(i)$, and denote the original position of the i^{th} element of the permutation as $\Omega^{-1}(i)$.*

Lemma 23 (Topological Reordering Rule): *Let $\mathbb{E} = (E_1, \cdots, E_n)$ be a Pauli exponential sequence with associated Pauli DAG $D = (V, A)$. For any permutation function Ω, if $T = \left(v_{\Omega^{-1}(1)}, \ldots, v_{\Omega^{-1}(n)}\right)$ is a topological ordering of D, the Pauli exponential sequence $\mathbb{E}' = \left(E_{\Omega^{-1}(1)}, \ldots, E_{\Omega^{-1}(n)}\right)$ satisfies $\mathbb{E} = \mathbb{E}'$.*

Proof of Lemma: We first define $\mathbb{E}'_i$ as having the same first i exponentials as $\mathbb{E}$ followed by the remaining elements in the same order as they appear in $\mathbb{E}'$.

We now aim to prove $\mathbb{E}'_i = \mathbb{E}'_{i+1}$. This proof is done by conceptually freezing the

first i Pauli exponentials of $\mathbb{E}'_i$ in place, then showing that E_{i+1} can be commuted to just after this *frozen* set. Let $B = \left\{v_j \in V | \Omega(j) < \Omega(i+1), j > i\right\}$, this is the set of vertices of D appearing before v_{i+1} in the topological ordering, ignoring any frozen vertices. Since T is a topological ordering of D, $\forall v_j \in B$, $v_{i+1} \notin \pi(v_j)$. By definition $(v_1, \ldots, v_n)$ is also a topological ordering of D, so we conclude $\forall v_j \in B$, $v_j \notin \pi(v_{i+1})$. I.e., by the definition of D they commute and $\forall v_j \in B$, $E_{i+1}E_j = E_jE_{i+1}$. We can then freely commute E_{i+1} to position $i+1$ of $\mathbb{E}'_i$, resulting in $\mathbb{E}'_{i+1}$.

By induction, we can conclude that $\mathbb{E}'_n = \mathbb{E}'_0$. It is trivial to see that $\mathbb{E} = \mathbb{E}'_n$ and $\mathbb{E}'_0 = \mathbb{E}'$, so we conclude that $\mathbb{E} = \mathbb{E}'$. ■

Using the Topological Reordering Rule, we can formally define the remaining rules we permit to apply to Pauli sequences. This second rule governs when we can use the Product Rotation Lemma to alter a Pauli exponential within a Pauli exponential sequence.

Lemma 24: *Let $\mathbb{E} = (E_1, \cdots, E_n)$ be a Pauli exponential sequence with associated Pauli DAG $D = (V, A)$, and $|\psi\rangle$ be a Clifford state. Consider some $E_i = \vec{P}(\alpha)$. Suppose $\pm\vec{Q}|\psi\rangle = |\psi\rangle$ for $\vec{Q}\vec{P} = \vec{P}\vec{Q}$ and there exists a permutation function Ω such that $T = \left(v_{\Omega^{-1}(1)}, \ldots, v_{\Omega^{-1}(n)}\right)$ is a topological ordering of D and*

$$\forall j : \Omega(j) < \Omega(i), E_j\vec{Q} = \vec{Q}E_j$$

Letting $\vec{R} = \vec{P}\vec{Q}$, we can create a Pauli exponential sequence

$$\mathbb{E}' = \left(E_{\Omega^{-1}(1)}, \ldots, E_{\Omega^{-1}(\Omega(i)-1)}, \vec{R}(\pm\alpha), E_{\Omega^{-1}(\Omega(i)+1)}, \ldots, E_{\Omega^{-1}(n)}\right)$$

Such that $\mathbb{E} \approx_{|\psi\rangle} \mathbb{E}'$.

Proof of Lemma: First, we define

$$\mathbb{E}_1 = \left(E_{\Omega^{-1}(1)}, \ldots, E_{\Omega^{-1}(n)}\right)$$

By the Topological Reordering Rule, $\mathbb{E} = \mathbb{E}_1$. Next, we define a state,

$$|\psi\rangle' = E_{\Omega^{-1}(\Omega(i)-1)} E_{\Omega^{-1}(\Omega(i)-2)} \cdots E_{\Omega^{-1}(1)} |\psi\rangle$$

We can now show that $\pm\vec{Q}$ is a stabiliser of $|\psi\rangle'$. To do this, we use our assumption about the commutation of $\vec{Q}$, $\forall j : \Omega(j) \leq \Omega(i), E_j \vec{Q} = \vec{Q} E_j$. Then, we use that $\pm\vec{Q}$ is a stabiliser of $|\psi\rangle$.

$$\begin{aligned}
\pm\vec{Q}|\psi\rangle' &= \pm\vec{Q} E_{\Omega^{-1}(\Omega(i)-1)} E_{\Omega^{-1}(\Omega(i)-2)} \cdots E_{\Omega^{-1}(1)} |\psi\rangle \\
&= \pm E_{\Omega^{-1}(\Omega(i)-1)} E_{\Omega^{-1}(\Omega(i)-2)} \cdots E_{\Omega^{-1}(1)} \vec{Q} |\psi\rangle \\
&= E_{\Omega^{-1}(\Omega(i)-1)} E_{\Omega^{-1}(\Omega(i)-2)} \cdots E_{\Omega^{-1}(1)} \left(\pm\vec{Q}|\psi\rangle\right) \\
&= E_{\Omega^{-1}(\Omega(i)-1)} E_{\Omega^{-1}(\Omega(i)-2)} \cdots E_{\Omega^{-1}(1)} |\psi\rangle \\
&= |\psi\rangle'
\end{aligned}$$

Since $\vec{P}\vec{Q} = \vec{Q}\vec{P}$, we can now apply the Product Rotation Lemma on $\vec{P}(\alpha)$ and $|\psi\rangle'$ to show that $C(\mathbb{E})|\psi\rangle = C(\mathbb{E}')|\psi\rangle$.

$$\begin{aligned}
C(\mathbb{E})|\psi\rangle &= C(\mathbb{E}_1)|\psi\rangle \\
&= E_{\Omega^{-1}(n)} \cdots E_{\Omega^{-1}(\Omega(i)+1)} \vec{P}(\alpha) E_{\Omega^{-1}(\Omega(i)-1)} E_{\Omega^{-1}(\Omega(i)-2)} \cdots E_{\Omega^{-1}(1)} |\psi\rangle \\
&= E_{\Omega^{-1}(n)} \cdots E_{\Omega^{-1}(\Omega(i)+1)} \left(\vec{P}(\alpha)|\psi\rangle'\right) \\
&= E_{\Omega^{-1}(n)} \cdots E_{\Omega^{-1}(\Omega(i)+1)} \left(\vec{R}(\pm\alpha)|\psi\rangle'\right) \\
&= E_{\Omega^{-1}(n)} \cdots E_{\Omega^{-1}(\Omega(i)+1)} \vec{R}(\pm\alpha) E_{\Omega^{-1}(\Omega(i)-1)} E_{\Omega^{-1}(\Omega(i)-2)} \cdots E_{\Omega^{-1}(1)} |\psi\rangle \\
&= C(\mathbb{E}')|\psi\rangle
\end{aligned}$$

We can now conclude that $\mathbb{E} \approx_{|\psi\rangle} \mathbb{E}'$, completing the proof. ■

The following rule governs when we can merge two Pauli exponentials with the same Pauli string with a Pauli exponential sequence.

Lemma 25: *Let $\mathbb{E} = (E_1, \cdots, E_n)$ be a Pauli exponential sequence with associated Pauli DAG $D = (V, A)$. Consider some $E_i = \vec{P}(\alpha), E_j = \vec{P}(\beta)$ with $i \neq j$. Suppose there exists a permutation function Ω such that $T = \left(v_{\Omega^{-1}(1)}, \ldots, v_{\Omega^{-1}(n)}\right)$ is a topological ordering of D and*

$$\Omega(i) + 1 = \Omega(j)$$

Then we can create a Pauli exponential sequence

$$\mathbb{E}' = \left(E_1, \ldots, E_{i-1}, \vec{P}(\alpha + \beta), E_{i+1}, \ldots, E_{j-1}, E_{j+1}, \ldots, E_n\right)$$

Such that $\mathbb{E} = \mathbb{E}'$.

Proof of Lemma: First, we define

$$\mathbb{E}_1 = \left(E_{\Omega^{-1}(1)}, \ldots, E_{\Omega^{-1}(\Omega(i)-1)}, \vec{P}(\alpha), \vec{P}(\beta), E_{\Omega^{-1}(\Omega(i)+2)}, \ldots, E_{\Omega^{-1}(n)}\right)$$

By the Topological Reordering Rule, $\mathbb{E} = \mathbb{E}_1$. Next, we define

$$\mathbb{E}_2 = \left(E_{\Omega^{-1}(1)}, \ldots, E_{\Omega^{-1}(\Omega(i)-1)}, \vec{P}(\alpha + \beta), E_{\Omega^{-1}(\Omega(i)+2)}, \ldots, E_{\Omega^{-1}(n)}\right)$$

We claim that $\mathbb{E}_1 = \mathbb{E}_2$ or equivalently, $C(\mathbb{E}_1) = C(\mathbb{E}_2)$

$$\begin{aligned}
C(\mathbb{E}_1) &= E_{\Omega^{-1}(n)} \cdots E_{\Omega^{-1}(\Omega(i)+2)} \vec{P}(\beta)\, \vec{P}(\alpha)\, E_{\Omega^{-1}(\Omega(i)-1)} \cdots E_{\Omega^{-1}(1)} \\
&= E_{\Omega^{-1}(n)} \cdots E_{\Omega^{-1}(\Omega(i)+2)} \vec{P}(\beta)\, \vec{P}(\alpha)\, E_{\Omega^{-1}(\Omega(i)-1)} \cdots E_{\Omega^{-1}(1)} \\
&= E_{\Omega^{-1}(n)} \cdots E_{\Omega^{-1}(\Omega(i)+2)} \vec{P}(\alpha + \beta)\, E_{\Omega^{-1}(\Omega(i)-1)} \cdots E_{\Omega^{-1}(1)} \\
&= C(\mathbb{E}_2)
\end{aligned}$$

Considering the Pauli DAG $D(\mathbb{E}_2)$, it is immediately clear that it contains the same vertices and edges as $D(\mathbb{E})$ but with v_j removed. As a result, we can conclude that $(v_1, \ldots, v_i, \ldots, v_{j-1}, v_{j+1}, \ldots, v_n)$ is a topological ordering of $D(\mathbb{E}_2)$. By the Topological Reordering Rule we see that $\mathbb{E}_2 = \mathbb{E}'$, and therefore $\mathbb{E} = \mathbb{E}'$. ■

As a consequence of these rules, we can define one final rule. This rule is the primary tool used throughout the report and allows for merging Pauli exponentials that initially do not have the same string.

Lemma 26: *Let $\mathbb{E} = (E_1, \cdots, E_n)$ be a Pauli exponential sequence with associated Pauli DAG $D = (V, A)$, and $|\psi\rangle$ be a Clifford state. Consider some $E_i = \vec{P}(\alpha), E_j = \vec{Q}(\beta)$ such that $\vec{P}\vec{Q} = \vec{Q}\vec{P}$. Suppose $\pm\vec{P}\vec{Q}|\psi\rangle = |\psi\rangle$ and there exists a permutation function Ω such that $T = \left(v_{\Omega^{-1}(1)}, \ldots, v_{\Omega^{-1}(n)}\right)$ is a topological ordering of D and we have both*

$$\Omega(i) + 1 = \Omega(j)$$

and

$$\forall k : \Omega(k) < \Omega(i), E_k\vec{P}\vec{Q} = \vec{P}\vec{Q}E_k$$

Then we can create a Pauli exponential sequence

$$\mathbb{E}' = \left(E_1, \ldots, E_{i-1}, \vec{P}(\alpha \pm \beta), E_{i+1}, \ldots, E_{j-1}, E_{j+1}, \ldots, E_n\right)$$

Such that $\mathbb{E} \approx_{|\psi\rangle} \mathbb{E}'$.

Proof of Lemma: First, we define

$$\mathbb{E}_1 = \left(E_{\Omega^{-1}(1)}, \ldots, E_{\Omega^{-1}(\Omega(i)-1)}, \vec{P}(\alpha), \vec{P}(\pm\beta), E_{\Omega^{-1}(\Omega(i)+1)}, \ldots, E_{\Omega^{-1}(n)}\right)$$

As $\vec{P}\left(\vec{P}\vec{Q}\right) = \vec{P}\left(\vec{Q}\vec{P}\right) = \left(\vec{P}\vec{Q}\right)\vec{P}$, and $\pm\vec{P}\vec{Q}\vec{Q} = \pm\vec{P}$ we satisfy the condi-

tions of Lemma 24 and conclude that $\mathbb{E} \approx_{|\psi\rangle} \mathbb{E}_1$. We can now define

$$\mathbb{E}_2 = \left(E_{\Omega^{-1}(1)}, \ldots, E_{\Omega^{-1}(\Omega(i)-1)}, \vec{P}\,(\alpha \pm \beta)\,, E_{\Omega^{-1}(\Omega(i)+2)}, \ldots, E_{\Omega^{-1}(n)}\right)$$

Satisfying the conditions of Lemma 25, it is easy to see that $\mathbb{E}_1 = \mathbb{E}_2$. Finally, by considering the Pauli DAG $D(\mathbb{E}_2)$, it is immediately clear that it contains the same vertices and edges as $D(\mathbb{E})$ but with v_j removed. As a result, we can conclude that $(v_1, \ldots, v_i, \ldots, v_{j-1}, v_{j+1}, \ldots, v_n)$ is a topological ordering of $D(\mathbb{E}_2)$. Therefore, by the Topological Reordering Rule we see that $\mathbb{E}_2 = \mathbb{E}'$. Putting this all together we get $\mathbb{E} \approx_{|\psi\rangle} \mathbb{E}_1 = \mathbb{E}_2 = \mathbb{E}'$, so $\mathbb{E} \approx_{|\psi\rangle} \mathbb{E}'$ concluding the proof. ■

We can actually see that Lemma 26 is a strict generalisation of Lemma 25 since $\vec{P}\vec{P} = \vec{I}$ is always a stabiliser and trivially commutes with all Pauli exponentials.

Ignoring the subsumed rule we are left with a set of rules, the Topological Reordering Rule, Lemma 24, and Lemma 26. The definitions of Lemma 24, Lemma 26 both use the existence of a topological ordering of a Pauli DAG that induces a favourable reordering of a Pauli exponential sequence. This notation is quite cumbersome, so we can instead define less generalised rules as follows.

Lemma 27 (Product Rotation Rule): *Let $\mathbb{E} = (E_1, \cdots, E_n)$ be a Pauli exponential sequence, and $|\psi\rangle$ be a Clifford state. Consider some $E_i = \vec{P}\,(\alpha)$. Suppose $\pm\vec{Q}|\psi\rangle = |\psi\rangle$ for $\vec{Q}\vec{P} = \vec{P}\vec{Q}$ and*

$$\forall j < i, E_j\vec{Q} = \vec{Q}E_j$$

Letting $\vec{R} = \vec{P}\vec{Q}$, we can create a Pauli exponential sequence

$$\mathbb{E}' = \left(E_1, \ldots, E_{i-1}, \vec{R}\,(\pm\alpha)\,, E_{i+1}, \ldots, E_n\right)$$

Such that $\mathbb{E} \approx_{|\psi\rangle} \mathbb{E}'$.

Proof of Lemma: $(v_1, \ldots, v_n)$ is a topological ordering satisfying the conditions of Lemma 24, so $\mathbb{E} \approx_{|\psi\rangle} \mathbb{E}'$. ■

Lemma 28 (Pauli Merge Rule): *Let* $\mathbb{E} = (E_1, \cdots, E_n)$ *be a Pauli exponential sequence with associated Pauli DAG* $D = (V, A)$*, and* $|\psi\rangle$ *be a Clifford state. Consider some* $E_i = \vec{P}(\alpha), E_j = \vec{Q}(\beta)$ *such that* $v_i \sim_D v_j$*. Suppose* $\pm\vec{P}\vec{Q}|\psi\rangle = |\psi\rangle$ *and*

$$\forall v_k \in \pi(v_i) \cup \pi(v_j), E_k\left(\vec{P}\vec{Q}\right) = \left(\vec{P}\vec{Q}\right) E_k$$

Letting $\vec{R} = \vec{P}\vec{Q}$*, we can create a Pauli exponential sequence*

$$\mathbb{E}' = \left(E_1, \ldots, E_{i-1}, \vec{R}(\alpha \pm \beta), E_{i+1}, \ldots, E_{j-1}, E_{j+1}, \ldots, E_n\right)$$

Such that $\mathbb{E} \approx_{|\psi\rangle} \mathbb{E}'$.

Proof of Lemma: We can place the elements of $\pi(v_i) \cup \pi(v_j)$ before v_i, then place v_j and finally the remaining elements after v_j to form a topological ordering of D. Since the members of $\pi(v_i) \cup \pi(v_j)$ are the only members before v_i and v_j in the topological ordering we satisfy the conditions for Lemma 26 so $\mathbb{E} \approx_{|\psi\rangle} \mathbb{E}'$. ■

This definition gives us the following corollary for trivial merges of Pauli exponentials.

Corollary 1 (Trivial Merge Rule): *Let* $\mathbb{E} = (E_1, \cdots, E_n)$ *be a Pauli exponential sequence with associated Pauli DAG* $D = (V, A)$*, and* $|\psi\rangle$ *be a Clifford state. Consider some* $E_i = \vec{P}(\alpha), E_j = \vec{P}(\beta)$ *such that* $v_i \sim_D v_j$

$$\mathbb{E}' = \left(E_1, \ldots, E_{i-1}, \vec{P}(\alpha \pm \beta), E_{i+1}, \ldots, E_{j-1}, E_{j+1}, \ldots, E_n\right)$$

Such that $\mathbb{E} \approx_{|\psi\rangle} \mathbb{E}'$.

Proof of Corollary: This follows directly from the Pauli Merge Rule using $\vec{I}$ as the merge string as it commutes with every Pauli string. ■

The conditions of the Pauli Merge Rule are precise and computationally expensive to compute. Every transitive parent of both Pauli exponentials must be checked for commutation. This motivated the Pauli DAG Merging Theorem, capturing the conditions for the Pauli Merge Rule in terms of ρ sets.

Theorem 2 (Pauli DAG Merging Theorem): *Let* $\mathbb{E} = (E_1, \cdots, E_n)$ *be a Pauli exponential sequence with associated Pauli DAG* $D = (V, A)$*, and* $|\psi\rangle$ *be a Clifford state. The Pauli Merge Rule can be applied to* $E_i = \vec{P}(\alpha)$ *and* $E_j = \vec{Q}(\beta)$ *iff* $\rho(v_i) = \rho(v_j)$ *and* $\pm\vec{P}\vec{Q}$ *is a stabiliser of* $|\psi\rangle$.

Proof of Theorem: $(\Rightarrow)$

If we can merge E_i with E_j through application of the Pauli Merge Rule then $v_i \sim_D v_j$ and

$$\forall v_k \in \pi(v_i) \cup \pi(v_j), E_k\left(\vec{P}\vec{Q}\right) = \left(\vec{P}\vec{Q}\right) E_k$$

Assume for the sake of contradiction that $\rho(v_i) \neq \rho(v_j)$. Then, without loss of generality, there exists $v_k \in \rho(v_i)$ such that $v_k \notin \rho(v_j)$. There are two cases to consider. First, where E_k commutes with $\vec{Q}$. In this case, we get the following.

$$\begin{aligned}\left(\vec{P}\vec{Q}\right)E_k &= \vec{P}\left(\vec{Q}E_k\right)\\ &= \vec{P}\left(E_k\vec{Q}\right)\\ &= \left(\vec{P}E_k\right)\vec{Q}\\ &= -E_k\left(\vec{P}\vec{Q}\right)\end{aligned}$$

This is a contradiction, so we are left with the case where E anti-commutes with $\vec{Q}$. Since $v_k \notin \rho\left(v_j\right)$, we have that $v_j \in \rho\left(v_k\right)$ and so $v_j \in \pi\left(v_k\right)$. However, since $\pi\left(v_k\right) \subset \pi\left(v_i\right)$, we have that $v_j \in \pi\left(v_i\right)$, contradicting $v_i \sim_D v_j$. We therefore conclude that $\rho\left(v_i\right) = \rho\left(v_j\right)$.

$(\Leftarrow)$

Since $\pm\vec{P}\vec{Q}|\psi\rangle = |\psi\rangle$, we need to show that $v_i \sim_D v_j$ and

$$\forall v_k \in \pi\left(v_i\right) \cup \pi\left(v_j\right), E_k\left(\vec{P}\vec{Q}\right) = \left(\vec{P}\vec{Q}\right)E_k$$

$v_i \sim_D v_j$ follows directly from $\rho\left(v_i\right) = \rho\left(v_j\right)$ as if there is a transitive path from v_i to v_j then there must exist some v_k such that $v_i \to v_k$ and $\left(v_k, v_j\right) \in A$. I.e., $v_k \notin \rho\left(v_i\right)$ and $v_k \in \rho\left(v_j\right)$, a contradiction.

Finally, we need to show

$$\forall v_k \in \pi\left(v_i\right) \cup \pi\left(v_j\right), E_k\left(\vec{P}\vec{Q}\right) = \left(\vec{P}\vec{Q}\right)E_k$$

by a case analysis on membership of $\rho\left(v_i\right)$ assuming $v_k \in \pi\left(v_i\right) \cup \pi\left(v_j\right)$.

Case 1: $v_k \in \rho(v_i)$

In this case, we get $v_k \in \rho(v_j)$. Using this we have $E_k\vec{P} = -\vec{P}E_k$ and $E_k\vec{Q} = -\vec{Q}E_k$. Finally, we get the following,

$$\begin{aligned} E_k\left(\vec{P}\vec{Q}\right) &= \left(E_k\vec{P}\right)\vec{Q} \\ &= -\left(\vec{P}E_k\right)\vec{Q} \\ &= -\vec{P}\left(E_k\vec{Q}\right) \\ &= \vec{P}\left(\vec{Q}E_k\right) \\ &= \left(\vec{P}\vec{Q}\right)E_k \end{aligned}$$

Case 2: $v_k \notin \rho(v_i)$

In this case, we get $v_k \notin \rho(v_j)$. Using this we have $E_k\vec{P} = \vec{P}E_k$ and $E_k\vec{Q} = \vec{Q}E_k$ as $v_k \in \pi(v_i) = \pi(v_j)$ so $k < j$ and $k < i$ and since $v_k \notin \rho(v_i) = \rho(v_j)$ they must commute. Finally, we get the following.

$$\begin{aligned} E_k\left(\vec{P}\vec{Q}\right) &= \left(E_k\vec{P}\right)\vec{Q} \\ &= \left(\vec{P}E_k\right)\vec{Q} \\ &= \vec{P}\left(E_k\vec{Q}\right) \\ &= \vec{P}\left(\vec{Q}E_k\right) \\ &= \left(\vec{P}\vec{Q}\right)E_k \end{aligned}$$

■

The Topological Reordering Rule, the Product Rotation Rule, and the Pauli Merge Rule form a sound ruleset for manipulating Pauli exponential sequences

that will be used for the remainder of the report. It is easy to see that the full rules of Lemma 24 and Lemma 26 can be recovered by applications of this simplified ruleset. We now wish to show that the Pauli Merge Rule is the only required rule for performing merges. In other words, rearranging the Pauli exponential sequence or changing Pauli strings using the Product Rotation Lemma cannot permit additional applications of the Pauli Merge Rule to a Pauli exponential sequence.

As a warmup, we prove that nothing can be gained from transforming both Pauli exponentials instead of just one of them. In other words, if two Pauli exponentials can be merged through the application of the Topological Reordering Rule, then the Product Rotation Rule and finally the Pauli Merge Rule, they can be merged by a single application of the Pauli Merge Rule.

Lemma 29: *Let* $\mathbb{E} = (E_1, \cdots, E_n)$ *be a Pauli exponential sequence with associated Pauli DAG* $D = (V, A)$*, and* $|\psi\rangle$ *be a Clifford state. Consider some* $E_i = \vec{P}(\alpha), E_j = \vec{Q}(\beta)$*. If* $\exists \vec{A}, \vec{B}$ *such that* $\vec{P}\vec{A} = \vec{A}\vec{P} = \vec{Q}\vec{B} = \vec{B}\vec{Q}$*,* $\vec{A}|\psi\rangle = \vec{B}|\psi\rangle = |\psi\rangle$*, and*

$$\forall v_k \in \pi(v_i) \cup \pi(v_j), E_k\vec{A} = \vec{A}E_k \wedge E_k\vec{B} = \vec{B}E_k$$

Then $\vec{P}\vec{Q}|\psi\rangle = |\psi\rangle$*,* $\vec{P}\vec{Q} = \vec{Q}\vec{P}$*, and*

$$\forall v_k \in \pi(v_i) \cup \pi(v_j), E_k\left(\vec{P}\vec{Q}\right) = \left(\vec{P}\vec{Q}\right)E_k$$

Proof of Lemma: First, we recognise that $\vec{P}\vec{Q} = \vec{A}\vec{B}$.

$$\begin{aligned}
\vec{A}\vec{P} = \vec{B}\vec{Q} &\implies \vec{A}\vec{A}\vec{P} = \vec{A}\vec{B}\vec{Q} \\
&\implies \vec{P} = \vec{A}\vec{B}\vec{Q} \\
&\implies \vec{P}\vec{Q} = \vec{A}\vec{B}\vec{Q}\vec{Q} \\
&\implies \vec{P}\vec{Q} = \vec{A}\vec{B}
\end{aligned}$$

Using this, we can prove that $\vec{P}\vec{Q}$ is a stabiliser of $|\psi\rangle$.

$$\begin{aligned}
\vec{P}\vec{Q}|\psi\rangle &= \vec{A}\vec{B}|\psi\rangle \\
&= \vec{A}|\psi\rangle \\
&= |\psi\rangle
\end{aligned}$$

We can use the facts that A and B must commute by Lemma 11, $\vec{P}\vec{Q} = \vec{A}\vec{B}$, and that $\vec{A}\vec{P} = \vec{B}\vec{Q}$ to show that $\vec{P}$ and $\vec{Q}$ must commute.

$$\begin{aligned}
\vec{P}\vec{Q} &= \vec{A}\vec{B} \\
&= \vec{B}\vec{A} \\
&= \vec{B}\vec{A}\vec{P}\vec{P} \\
&= \vec{B}\vec{B}\vec{Q}\vec{P} \\
&= \vec{Q}\vec{P}
\end{aligned}$$

Finally, $\forall v_k \in \pi(v_i) \cup \pi(v_j)$ we have the following.

$$\begin{aligned} E_k\left(\vec{P}\vec{Q}\right) &= E_k\vec{A}\vec{B} \\ &= \vec{A}E_k\vec{B} \\ &= \vec{A}\vec{B}E_k \\ &= \left(\vec{P}\vec{Q}\right)E_k \end{aligned}$$

Concluding the proof. ■

Before demonstrating the full proof, we present an instrumental lemma characterising the changes in ρ sets by applications of the Product Rotation Rule.

Lemma 30: *Let $\mathbb{E} = (E_1, \cdots, E_n)$ be a Pauli exponential sequence with associated Pauli DAG $D = (V, A)$, and $|\psi\rangle$ be a Clifford state. Fix some $E_i = \vec{P}(\alpha) \in \mathbb{E}$ that satisfies the conditions of Product Rotation Rule. Let $\mathbb{E}'$ be the Pauli exponential sequence with associated Pauli DAG $D' = (V, A')$ after applying the Product Rotation Rule to E_i with stabiliser $\vec{S}$. Then, $\forall v_k \in V$ with $E_k = \overrightarrow{Q_k}(\beta)$,*

$$\rho_{D'}(v_k) = \begin{cases} \rho_D(v_k) & \text{where } \overrightarrow{Q_k}\vec{S} = \vec{S}\overrightarrow{Q_k} \\ \rho_D(v_k) \triangle \{v_i\} & \text{otherwise} \end{cases}$$

Where $\triangle$ is the symmetric difference operator.

Proof of Lemma: Fix an arbitrary vertex $v_k \in V$. Since only E_i changed, only edges incident to v_i can be affected. After the application, E_i now has the Pauli string $\vec{P}\vec{S}$.

Case 1: $\overrightarrow{Q_k}\vec{S} = \vec{S}\overrightarrow{Q_k}$

In this case, if $\overrightarrow{Q_k}$ commutes with $\vec{P}$ then,

$$\left(\vec{P}\vec{S}\right)\overrightarrow{Q_k} = \vec{P}\overrightarrow{Q_k}\vec{S} = \overrightarrow{Q_k}\left(\vec{P}\vec{S}\right)$$

So there is still no edge between v_k and v_i in D'. If instead $\overrightarrow{Q_k}$ anti-commutes with $\vec{P}$ then,

$$\left(\vec{P}\,\vec{S}\right)\overrightarrow{Q_k} = \vec{P}\overrightarrow{Q_k}\vec{S} = -\overrightarrow{Q_k}\left(\vec{P}\,\vec{S}\right)$$

So the edge between v_k and v_i remains in D'. Therefore, we can conclude that

$$\rho_{D'}\left(v_k\right) = \rho_D\left(v_k\right)$$

Case 2: $\overrightarrow{Q_k}\,\vec{S} = -\,\vec{S}\,\overrightarrow{Q_k}$

In this case, we first observe that E_i appears before E_k in $\mathbb{E}$. This is a requirement of the Product Rotation Rule since the stabiliser does not commute with $\overrightarrow{Q_k}$. Therefore, if the edge between v_i and v_k is updated, it will only affect the ρ set of v_k. First, we assume that $\overrightarrow{Q_k}$ commutes with $\vec{P}$ then,

$$\left(\vec{P}\,\vec{S}\right)\overrightarrow{Q_k} = -\vec{P}\overrightarrow{Q_k}\vec{S} = -\overrightarrow{Q_k}\left(\vec{P}\,\vec{S}\right)$$

This toggles the edge between v_i and v_k. Next, we assume that $\overrightarrow{Q_k}$ anti-commutes with $\vec{P}$ then,

$$\left(\vec{P}\,\vec{S}\right)\overrightarrow{Q_k} = -\vec{P}\overrightarrow{Q_k}\vec{S} = \overrightarrow{Q_k}\left(\vec{P}\,\vec{S}\right)$$

This again toggles the edge between v_i and v_k, allowing us to conclude that

$$\rho_{D'}\left(v_k\right) = \rho_D\left(v_k\right) \triangle \{v_i\}$$

■

Finally, we can prove that the Pauli Merge Rule is sufficient for performing merges in Pauli exponential sequences.

Theorem 3: *Let $\mathbb{E} = (E_1, \cdots, E_n)$ be a Pauli exponential sequence with associated Pauli DAG $D = (V, A)$, and $|\psi\rangle$ be a Clifford state. Using only the Topological Reordering Rule, the Product Rotation Rule, and the Pauli Merge Rule, two Pauli exponentials in the sequence can be merged iff they can be merged by a single application of the Pauli Merge Rule.*

Proof of Theorem: $(\Leftarrow)$

Since the Pauli Merge Rule is one of the rules, this direction is trivial.

$(\Rightarrow)$

First, we fix two Pauli exponentials $E_i = \vec{P}(\alpha)$ and $E_j = \vec{Q}(\alpha)$ and consider a sequence of rule applications $\mathbb{R} = (R_1, \ldots, R_n)$. such that R_n is an application of the Pauli Merge Rule to merge E_i and E_j. To simplify this sequence, we can assume without loss of generality, that applications of the Pauli Merge Rule within the sequence (excluding R_n) are all trivial merges by the Trivial Merge Rule. This can be done without loss of generality by observing that the Pauli Merge Rule is the composition of the Product Rotation Rule and a trivial merge using Trivial Merge Rule.

We first perform a forward induction on $\mathbb{R}$ to show that after the $(n-1)^{th}$ rule is applied, any Pauli exponential $E_k = \vec{T}(\gamma)$ not removed by the Pauli Merge Rule is of the form $\vec{T'}(\pm\gamma + \delta)$ with $\vec{T'} = \vec{T}\vec{S}$ where $\vec{S}$ is a stabiliser of $|\psi\rangle$, $\vec{S}\vec{T} = \vec{T}\vec{S}$, and δ is a constant.

After no rules have been applied, this is trivially true by choosing $\vec{S} = \vec{I}$ and $\delta = 0$. For the inductive step, we have $E_k = \vec{T'}(\pm\gamma + \delta)$. First, we observe that, since the Topological Reordering Rule only changes ordering, it will not affect any Pauli exponentials.

Next, applications of the Product Rotation Rule to E_k using stabiliser $\vec{S'}$ will give $E_k = \overrightarrow{T''}(\pm\gamma + \delta')$ where $\overrightarrow{T''} = \vec{T}\vec{S}\vec{S'}$ and $\delta' = \pm\delta$. Since $\vec{S}$ and $\vec{S'}$ are both stabilisers, they combine to create a new stabiliser $\overrightarrow{S''}$. As $\vec{S}$ and $\vec{S'}$ both commute with $\vec{T}$, so does $\overrightarrow{S''}$. This gives $\overrightarrow{T''} = \vec{T}\overrightarrow{S''}$, giving the Pauli exponential the desired form.

Finally, we know by the assumption that no rule in the first $(n-1)$ rules of $\mathbb{R}$ can merge E_k into a different Pauli exponential. Therefore, trivial merges into E_k will give $E_k = \vec{T'}(\pm\gamma + \delta')$ with $\delta' = \delta + \omega$ where ω is the phase of the Pauli exponential merged into E_k. This is of the desired form.

Having shown the forward induction, we can use it to show that the Pauli Merge Rule rule applies to E_i and E_j before the other rules are applied. We do this by using the conditions of Pauli DAG Merging Theorem. Therefore, we need to show that $\pm\vec{P}\vec{Q}$ is a stabiliser of $|\psi\rangle$ and $\rho_D(v_i) = \rho_D(v_j)$.

As R_n merges E_i and E_j using the Pauli Merge Rule, neither can have been merged into another Pauli exponential. Therefore, after the $(n-1)^{th}$ rule has been applied, $E_i = \vec{P'}(\alpha \pm \delta_1)$ and $E_j = \vec{Q'}(\beta \pm \delta_2)$ with $\vec{P'} = \vec{P}\vec{S_1}$ and $\vec{Q'} = \vec{Q}\vec{S_2}$.

As the rule R_n merges E_i and E_j using the Pauli Merge Rule, we know that $\vec{P}\vec{S_1}\vec{Q}\vec{S_2}$ is a stabiliser of $|\psi\rangle$. As products of stabilisers are also stabilisers and $\vec{P}$ commutes with $\vec{S_1}$ then,

$$\begin{aligned} \pm\vec{P}\vec{S_1}\vec{Q}\vec{S_2}|\psi\rangle = |\psi\rangle &\implies \pm\vec{S_1}\vec{P}\vec{Q}\vec{S_2}|\psi\rangle = |\psi\rangle \\ &\implies \pm\vec{S_1}\vec{S_1}\vec{P}\vec{Q}\vec{S_2}\vec{S_2}|\psi\rangle = |\psi\rangle \\ &\implies \pm\vec{P}\vec{Q}|\psi\rangle = |\psi\rangle \end{aligned}$$

Finally, we need to show that $\rho_D(v_i) = \rho_D(v_j)$. To do this, we first define $D_k = D(\mathbb{E}_k)$ where $\mathbb{E}_k$ is $\mathbb{E}$ after the first k rules in $\mathbb{R}$ have been applied and $\mathbb{E}_0 = \mathbb{E}$. We now show by backwards induction that $\forall l \in [0, n]$, $\rho_{D_l}(v_i) = \rho_{D_l}(v_j)$. We begin with $l = n$. Since R_n merges E_i and E_j under $\mathbb{E}_n$, $\rho_{D_n}(v_i) = \rho_{D_n}(v_j)$ by the Pauli DAG Merging Theorem.

Next, we consider the case where $l < n$ by assuming it is true for $l + 1$, breaking it down into a case for each rule.

Case 1: Applying the Topological Reordering Rule.

Since these operations only permit topological orderings, if,

$$\rho_{D_{l+1}}(v_i) = \rho_{D_{l+1}}(v_j)$$

Then,

$$\rho_{D_l}(v_i) = \rho_{D_l}(v_j)$$

Case 2: Applying the Product Rotation Rule.

Let $\vec{T}$ be the Pauli stabiliser used during the application of the rule. Then we can prove that $\vec{T}$ commutes with $\vec{P}\vec{S_1}$ if and only if $\vec{T}$ commutes with $\vec{Q}\vec{S_2}$. Without loss of generality, we assume that $\vec{T}$ commutes with $\vec{P}\vec{S_1}$. As R_n can be applied after rule $l + 1$, then $\vec{P}\vec{S_1}\vec{Q}\vec{S_2}$ is a stabiliser of $|\psi\rangle$. Then, using Lemma 11 to observe that $\left(\vec{P}\vec{S_1}\vec{Q}\vec{S_2}\right)\vec{T} = \vec{T}\left(\vec{P}\vec{S_1}\vec{Q}\vec{S_2}\right)$ we get,

$$\begin{aligned}
\vec{T}\left(\vec{Q}\vec{S_2}\right) &= \vec{T}\left(\vec{P}\vec{S_1}\right)\left(\vec{P}\vec{S_1}\right)\vec{Q}\vec{S_2} \\
&= \left(\vec{P}\vec{S_1}\right)\vec{T}\left(\vec{P}\vec{S_1}\vec{Q}\vec{S_2}\right) \\
&= \left(\vec{P}\vec{S_1}\right)\left(\vec{P}\vec{S_1}\vec{Q}\vec{S_2}\right)\vec{T} \\
&= \left(\vec{Q}\vec{S_2}\right)\vec{T}
\end{aligned}$$

As $\vec{T}$ commutes with $\vec{P}\vec{S_1}$ if and only if $\vec{T}$ commutes with $\vec{Q}\vec{S_2}$, when $\vec{T}$ is applied to any Pauli exponential, by Lemma 30 their ρ sets undergo the same update. Therefore, $\rho_{D_l}(v_i) = \rho_{D_l}(v_j)$.

Case 3: Applying the Pauli Merge Rule to perform a trivial merge.

Since the merge is trivial, only one vertex v_a is removed by merging E_a with some E_b. As E_a is merged with E_b, v_a and v_b have identical properties in D_{l+1}. Therefore, as $\rho_{D_{l+1}}(v_i) = \rho_{D_{l+1}}(v_j)$, we get,

$$\begin{aligned} v_a \in \rho_{D_l}(v_i) &\Leftrightarrow v_b \in \rho_{D_l}(v_i) \\ &\Leftrightarrow v_b \in \rho_{D_{l+1}}(v_i) = \rho_{D_{l+1}}(v_j) \\ &\Leftrightarrow v_b \in \rho_{D_l}(v_j) \\ &\Leftrightarrow v_a \in \rho_{D_l}(v_j) \end{aligned}$$

We conclude $\rho_{D_l}(v_i) = \rho_{D_l}(v_j)$

By induction, this shows that $\rho_{D_n}(v_i) = \rho_{D_n}(v_j) \implies \rho_{D_0}(v_i) = \rho_{D_0}(v_j) \implies \rho_D(v_i) = \rho_D(v_j)$. This covers all conditions of the Pauli Merge Rule as characterised by the Pauli DAG Merging Theorem, concluding the proof. ■

The main implication of this proof is that the order in which merges are performed does not affect which merges are possible. This means, assuming we have checked the conditions of the Pauli Merge Rule for two Pauli exponentials E_i, E_j, if we only perform operations from the ruleset the Topological Reordering Rule, the Product Rotation Rule, and the Pauli Merge Rule, any subsequent checks of the conditions will not lead to a merge.

As a consequence of the Pauli DAG Merging Theorem, we can also easily prove

that merging within a Pauli exponential sequence by the Pauli Merge Rule is transitive.

Theorem 4: *Let $\mathbb{E} = (E_1, \cdots, E_n)$ be a Pauli exponential sequence with associated Pauli DAG $D = (V, A)$, and $|\psi\rangle$ be a Clifford state. Let $E_i = \vec{P}(\alpha), E_j = \vec{Q}(\beta), E_k = \vec{R}(\gamma) \in \mathbb{E}$ such that applications of the Pauli Merge Rule can merge E_i with E_j and E_j with E_k. Then we can conclude that the pair E_i and E_k satisfy the conditions for the Pauli Merge Rule.*

Proof of Theorem: As E_i can merge with E_j, we have that $\rho(v_i) = \rho(v_j)$. Since E_j can merge with E_k, we can conclude that $\rho(v_i) = \rho(v_j) = \rho(v_k)$.

As E_i can merge with E_j, we have that $\vec{P}\vec{Q}$ is a stabiliser of $|\psi\rangle$. Since E_j can merge with E_k, we can conclude that $\vec{Q}\vec{R}$ is also a stabiliser. Since the product of any two stabilisers is also a stabiliser, $\vec{P}\vec{Q}\vec{Q}\vec{R} = \vec{P}\vec{R}$ is a stabiliser. This satisfies the conditions of the Pauli DAG Merging Theorem so the pair E_i and E_k satisfy the conditions for the Pauli Merge Rule. ■

It also gives us a convenient tool for reasoning about cancelling Paulis in Pauli exponential sequences.

Lemma 31: *Let $\mathbb{E} = (E_1, \cdots, E_n)$ be a Pauli exponential sequence with associated Pauli DAG $D = (V, A)$, and $|\psi\rangle$ be a Clifford state. Let $E_j = \vec{P}(\alpha) \in \mathbb{E}$. E_j can be cancelled from the Pauli DAG by the rules the Topological Reordering Rule, the Product Rotation Rule, and the Pauli Merge Rule iff, $\rho(v_i) = \emptyset$ and $\vec{P}$ is a stabiliser of $|\psi\rangle$.*

Proof of Lemma: Cancelling is equivalent to merging with the identity Pauli $\vec{I}$. Therefore, we can consider all Pauli exponential sequences as having an identity element $E_I = \vec{I}(0)$. This identity element does not change the meaning of the circuit, so we can do so without loss of generality. Since $\vec{I}$ commutes with any Pauli string, $\rho\left(\vec{I}\right) = \emptyset$. By the Pauli DAG Merging Theorem we therefore

conclude E_j can be cancelled iff $\rho\left(\vec{P}\right) = \emptyset$ and $\vec{P}\,\vec{I} = \vec{P}$ is a stabiliser. ■

Finally, we get the following lemma of the number of times we need to perform cancellations.

Lemma 32: *Let $\mathbb{E} = (E_1, \cdots, E_n)$ be a Pauli exponential sequence with associated Pauli DAG $D = (V, A)$, and $|\psi\rangle$ be a Clifford state. After exhaustive applications of the Pauli Merge Rule, at most a single Pauli exponential in $\mathbb{E}$ can be cancelled.*

Proof of Lemma: If no Pauli exponentials can be cancelled initially, we are done. Otherwise, this follows directly Lemma 31. Every Pauli exponential $E_i = \vec{P}(\alpha)$ that could initially be cancelled has $\rho(v_j) = \emptyset$ and $\vec{P}$ is a stabiliser of $|\psi\rangle$. These all satisfy the conditions of the Pauli DAG Merging Theorem. Therefore, exhaustively applying the Pauli Merge Rule results in a single Pauli exponential that can be cancelled. ■

3.4 Reducing Pauli Strings

Once we wish to synthesise our Pauli DAG into a circuit, the number of Clifford gates in the synthesised circuit relies on the Pauli string in the Pauli exponentials being synthesised. Therefore, we can apply the Product Rotation Lemma before synthesis to reduce the number of synthesised Cliffords. As part of the synthesis, it makes sense to do so in mutually commuting groups. We need the following lemma to maintain the groups while applying the Product Rotation Lemma.

Lemma 33: *Let $\mathbb{E} = (E_1, \cdots, E_n)$ be a Pauli exponential sequence such that for any $E_i, E_j \in \mathbb{E}$, $E_iE_j = E_jE_i$, and $|\psi\rangle$ be a Clifford state. Applications of the Product Rotation Rule maintain the mutually commuting property iff the stabiliser applied commutes with all members of $\mathbb{E}$.*

Proof of Lemma: As every Pauli exponential commutes, any permutation of $\mathbb{E}$, $\mathbb{E}'$ satisfies $C(\mathbb{E}) = C(\mathbb{E}')$.

$(\Rightarrow)$

Without loss of generality we can apply the Product Rotation Lemma to $E_1 = \vec{P}(\alpha)$. Let $\vec{S}$ be a stabiliser of $|\psi\rangle$, such that $\vec{S}\vec{P} = \vec{P}\vec{S}$ and $\forall E_i = \vec{Q}(\beta) \in \mathbb{E}$, $\left(\vec{P}\vec{S}\right)\vec{Q} = \vec{Q}\left(\vec{P}\vec{S}\right)$. Fix any $E_i = \vec{Q}(\beta) \in \mathbb{E}$, we get the following.

$$\begin{aligned}
\vec{Q}\vec{S} &= \vec{Q}\vec{P}\vec{P}\vec{S} \\
&= \left(\vec{Q}\vec{P}\right)\left(\vec{P}\vec{S}\right) \\
&= \left(\vec{P}\vec{Q}\right)\left(\vec{P}\vec{S}\right) \\
&= \vec{P}\left(\vec{Q}\left(\vec{P}\vec{S}\right)\right) \\
&= \vec{P}\left(\left(\vec{P}\vec{S}\right)\vec{Q}\right) \\
&= \left(\vec{P}\vec{P}\right)\left(\vec{S}\vec{Q}\right) \\
&= \vec{I}\left(\vec{S}\vec{Q}\right) \\
&= \vec{S}\vec{Q}
\end{aligned}$$

$(\Leftarrow)$

Let $\vec{S}$ be a stabiliser of $|\psi\rangle$ such that $\forall E_i = \vec{P}(\alpha) \in \mathbb{E}$, $\vec{S}\vec{P} = \vec{P}\vec{S}$. Fix any two $E_i = \vec{P}(\alpha), E_j = \vec{Q}(\beta) \in \mathbb{E}$. We can consider any permutation $\mathbb{E}'$ of $\mathbb{E}$ such that $\vec{P}$ appears at the start. In this case, $\mathbb{E}'$ is directly after $|\psi\rangle$ so we can apply the Product Rotation Lemma resulting in $\vec{P}\vec{S}$. Finally, we must show that the new

Pauli commutes with $\vec{Q}$.

$$\begin{aligned}\vec{Q}\left(\vec{P}\,\vec{S}\right) &= \left(\vec{Q}\vec{P}\right)\vec{S} \\ &= \left(\vec{P}\vec{Q}\right)\vec{S} \\ &= \vec{P}\left(\vec{Q}\vec{S}\right) \\ &= \vec{P}\left(\vec{S}\vec{Q}\right) \\ &= \left(\vec{P}\,\vec{S}\right)\vec{Q}\end{aligned}$$

■

Considering Clifford states, we represent the stabilisers using q Pauli stabiliser generators. Therefore, we need to be able to find a new generating set of Pauli stabilisers whose group always commutes with an individual Pauli exponential. Using this observation, we must find Pauli stabiliser generators that commute with an entire group.

Lemma 34: *Consider a set of q Pauli stabiliser generators* $\overrightarrow{S_1}, ..., \overrightarrow{S_n}$ *and a Pauli exponential* $\vec{P}(\alpha)$ *such that at least one member of the group spanned by the generators does not commute with* $\vec{P}(\alpha)$*. There is a new set of Pauli stabiliser generators* $\overrightarrow{Q_1}, ..., \overrightarrow{Q_{q-1}}$ *whose group contains all members of the original group that commute with* $\vec{P}(\alpha)$*.*

Proof of Lemma: Let A be the subset of stabiliser generators that do not commute with $\vec{P}(\alpha)$. Since at least one group member does not commute with $\vec{P}$, $|A| \geq 1$.

First, we can look at the case where $|A| = 1, A = \left\{\vec{Q}\right\}$. We show that any product of the stabiliser generators of the group that includes $\vec{Q}$ does not commute with $\vec{P}(\alpha)$. If this is true and all members without $\vec{Q}$ do commute with $\vec{P}(\alpha)$, then $\vec{Q}$ can be removed from the generators. Let $\vec{S} = \vec{S_i} \cdots \vec{S_j}$ where

$\vec{S_k} \notin A$ be an arbitrary member of the group without the generator $\vec{Q}$.

$$\begin{aligned}\vec{P}\left(\vec{S}\vec{Q}\right) &= \vec{P}\vec{S_i}\cdots\vec{S_j}\vec{Q} \\ &= \vec{S_i}\cdots\vec{S_j}\vec{P}\vec{Q} \\ &= -\vec{S_i}\cdots\vec{S_j}\vec{Q}\vec{P} \\ &= -\left(\vec{S}\vec{Q}\right)\vec{P}\end{aligned}$$

$$\begin{aligned}\vec{P}\vec{S} &= \vec{P}S_i\cdots S_j \\ &= \vec{S_i}\cdots\vec{S_j}\vec{P} \\ &= \vec{S}\vec{P}\end{aligned}$$

This confirms that our new group satisfies the conditions. For the case where $|A| > 1, A = \left\{\overrightarrow{A_0}, ..., \overrightarrow{A_k}\right\}$ we recognise that the product of any two members of A gives a Pauli string that commutes with $\vec{P}$. Using this knowledge, we construct a new generating group as follows.

$$G = \vec{S_i}, ..., \vec{S_j}, \overrightarrow{A_0}\overrightarrow{A_1}, ..., \overrightarrow{A_0}\overrightarrow{A_k}, \overrightarrow{A_0}$$

We can observe that the group generated by G is the same as the original group, as all generators can be recovered. However, in this new group, all pairs from $A = \left\{\overrightarrow{A_0}, ..., \overrightarrow{A_k}\right\}$ now commute with $\vec{P}$ so we are back to the original case where $|A| = 1, A = \left\{\overrightarrow{A_0}\right\}$. ■

The consequence of this lemma is that it gives rise to an algorithm where we can access a generating set of the stabiliser group of a Clifford state *after* a series of Pauli exponentials. This generating set allows us to repeatedly apply the Product Rotation Lemma to change the Pauli exponentials in later groupings

without checking commutativity with a series of Pauli exponentials for every applied Pauli stabiliser.

Chapter 4

Design

The goal of this project is not only to investigate the theory behind applying the Product Rotation Lemma in the context of general circuits but also to demonstrate how practical this is as a technique. A software design is presented here with implementation details given in Chapter 5 of an end-to-end compiler to demonstrate the effectiveness and practicality of the technique. A common circuit representation is Quantum Assembly (QASM), a low-level assembly-like language for describing Quantum circuits. The most common implementation of QASM is in the form of OpenQASM, a continually updated open-source standard created by Qiskit [9]. This compiler takes QASM as input and outputs optimised QASM. The results of this software system are shown in chapter Chapter 6. The decision to design an end-to-end compiler allows us to see the practical applications of this method within a complete optimisation pipeline. It is hard to say what the new technique adds without other components to the compiler. However, with additional methods, it is easy to toggle on and off different elements to observe the benefits gained. We choose this style of a QASM-to-QASM compiler to allow a wide range of benchmark circuits and

applications to all circuits.

It is crucial to keep the metrics we are looking to optimise in mind during the design of a system. The most important metric to consider is the non-Clifford count. This metric is the number of non-Clifford gates that appear in the circuit after re-synthesis. The next most impactful metric we will consider is the final circuit's two-qubit count. For our context, this is the number of CX gates. Alongside the *number* of each type of gate, we also wish to minimise the *depth* required in the final circuit of non-Clifford and CX gates. Finally, we want to minimise the circuit's total gate number and total gate depth. The motivation for these three metrics comes from the cost of implementing each type of gate and the time taken to apply gates sequentially. Implementing a non-Clifford rotation is several orders of magnitude more expensive than any Clifford gate under the fault-tolerant model [22]. Implementing a CX gate is an order of magnitude more costly than a single qubit gate [19].

4.1 Internal Representation

A vital aspect of the optimisation procedure is its internal representation. For this, we need two data structures, one to handle the Pauli DAG for the circuit and the other to process the Cliffords. A suitable choice for this Clifford component is a unitary tableau while the Cliffords are moving through the circuit and a stabiliser-destabiliser tableau for describing the Clifford as a state [1]. The unitary tableau is a space-efficient data structure that defines a Clifford unitary entirely by tracking an input and output set of Pauli strings. The stabiliser-destabiliser tableau tracks the stabilisers of the $|0\rangle^{\otimes q}$ state through a series of tableau updates leading to a new generating set of stabilisers for a full state. The

destabilisers of a state are a set of Pauli generators that complete the group P_q. By the Gottesman-Knill theorem [14], the stabilisers of a Clifford state uniquely identify it, allowing us to reconstruct the Clifford state during synthesis [1, 24]. As a result of these choices, we refer to the internal representation as the Tableau-Pauli-DAG form.

4.2 The Pipeline

As mentioned previously, the input to the system will be in the QASM format. From here, the circuit needs converting to Tableau-Pauli-DAG; optimisations are applied before the final QASM is synthesised. The main steps of the procedure are outlined below.

- Conversion to Tableau-Pauli-DAG form
- Repeated application of the Pauli DAG Merging Theorem to the Pauli DAG
- Optimising the Pauli DAG further by recognising Clifford Pauli exponentials
- Pauli Exponential Synthesis
  - Grouping into mutually commuting sets of Pauli exponentials
  - Reduction of Pauli strings within mutually commuting sets of Pauli exponentials
  - Diagonalisation of mutually commuting groups
  - Synthesising remaining Phase polynomials

- Clifford Synthesis
  - Stabiliser-destabiliser tableau synthesis
  - Translating Cliffords to Pauli exponentials to apply Pauli DAG merging optimisations
  - Final optimised Clifford synthesis

4.3 Tableau-Pauli-DAG Form

Initially, we need to convert the input into a processable form, requiring the transformation of the circuit using Theorem 1. This transformation will result in a list of Clifford gates and a series of Pauli exponentials. I.e., the circuit will be in Clifford-Pauli-Exponential form. From this point, we need to convert the sequence of Pauli exponentials into a Pauli DAG and use both the stabiliser-destabiliser tableau and unitary tableau data structures to store the Cliffords.

4.3.1 Stabiliser-Destabiliser Tableau

Scott Aaronson and Daniel Gottesman present the stabiliser-destabiliser tableau algorithm in [1]. The complete algorithm is presented as a series of black box operations, so we give an alternative presentation here.

We begin with an initial state. By convention, we start with the state $|\psi\rangle = |0\rangle^{\otimes q}$. Applying a Z Pauli to each qubit will not change the resulting state, so we initially have q state stabilisers. By the definition of stabilisers, the product of any two stabilisers is also a stabiliser. As the q stabilisers for the circuit are linearly independent, we can conclude we have a group of 2^q stabilisers. Since

every Clifford state has exactly 2^q stabilisers, we conclude that we know all stabilisers of the circuit.

We can now consider applying a Clifford C to the state and attempting to find the stabiliser of the new state.

$$\begin{aligned} C|\psi\rangle &= C\vec{S}|\psi\rangle \\ &= C\vec{S}C^\dagger C|\psi\rangle \\ &= \left(C\vec{S}C^\dagger\right)C|\psi\rangle \end{aligned}$$

This equation tells us that for a stabiliser $\vec{S}$ of the state, we can apply any Clifford to the state and obtain an updated stabiliser simply by conjugating the original stabiliser by the Clifford. As demonstrated in Lemma 18, Lemma 19, and Lemma 20, where $\vec{S}$ is a Pauli string, this is easy to calculate. We can perform this operation for every initial stabiliser generator until we are left with a new set of generators. We know this new set of generators must also span a group of size 2^q since linear independence is preserved.

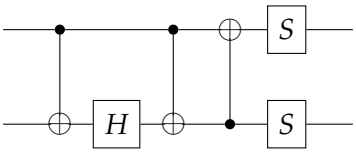
As an example, consider the two-qubit system initialised by the $|00\rangle$ state. This state initially has the stabiliser generators ZI and IZ. We can apply the gate $CX_{1,2}$ to this state. Using Lemma 20, this transforms the stabiliser generators such that ZZ and IZ are the new generators. Then, applying H_2 to the state changes the stabilisers such that the new generators are now ZX and IX.

We can look at a worst-case scenario where we have $\mathcal{O}(n)$ Pauli exponentials that need to be conjugated by $\mathcal{O}(n)$ Clifford gates. Each conjugation takes constant time, assuming the Pauli string has constant time random access. The

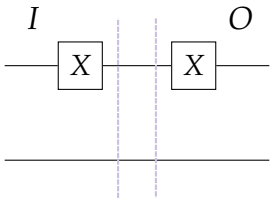
cost of including a single Clifford gate into the tableau takes time $\mathcal{O}(q)$, where q is the number of qubits. Therefore, including $\mathcal{O}(n)$ Clifford gates into the tableau takes time $\mathcal{O}(q \cdot n)$. Overall, this gives the procedure a time complexity of $\mathcal{O}\left(n^2 + q \cdot n\right)$, quadratic in the number of gates in the circuit. To counter this, we can make use of a unitary tableau.

4.3.2 Unitary Tableau

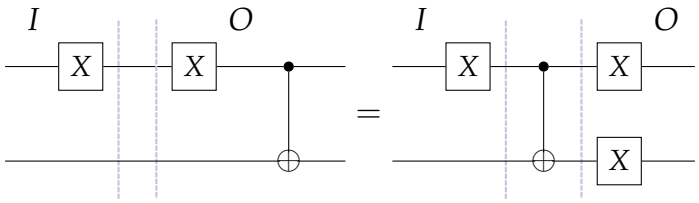
Unlike a state tableau, where we only wish to know what happens when conjugating the stabilisers of the original state, for a unitary tableau, we need to know the output of conjugating any Pauli string. This can be achieved by taking an ordered set of generators for the full Pauli group over the inputs and giving the action on each of these in order, allowing combinations by Lemma 17. By unitarity, these actions are generators of the full Pauli group over the outputs, so we may refer to them as the Z_i and X_i generators. The idea is that conjugating the input Pauli strings through the Clifford circuit results in the associated output Pauli strings (and vice versa). To clarify this, we can work through an example over the two-qubit system to compute the unitary tableau for the following circuit.



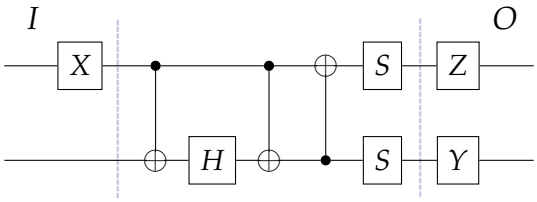
Initially, we start with the empty circuit; for simplicity, we will focus on the X_1 generators, i.e., where the input and output generators are initially both X_1. We split the diagram by the input and output generators, labelled I and O, respectively.



Our goal is to fix the input generators and find the output generators, so we assume this tableau appears at the start of the circuit. Therefore, we wish to push each gate in the circuit into the central portion by conjugation with the outputs. We consider the first CX gate and apply Lemma 20 to pull it into the middle.



We can repeat this for the remaining gates in the circuit using Lemma 18, Lemma 19, and Lemma 20 to give this final form.



From this equation, we conclude that the output stabiliser for the X_1 input stabiliser is ZY. We can repeat this process for each input stabiliser giving the following table, the unitary tableau for the circuit[1]. We include the signs produced during the process in the tableau for usage during the Product Rotation Lemma applications.

[1]We can note that the output generators for the Z_1 and Z_2 correspond to some choice of stabiliser generators, and output generators for the X_1 and X_2 to some choice of destabiliser generators of the circuit applied to the $|0\rangle^{\otimes q}$ state

Input	Output
X_1	$+ZY$
X_2	$+ZI$
Z_1	$+ZZ$
Z_2	$-XX$

To ensure that the input stabilisers remain unchanged, we note that Cliffords can only be included by conjugation with the outputs. However, the algorithm pushes the unitary tableau from the end of the circuit to the start, accumulating Cliffords as it goes. Therefore, we need a clear definition of how to push Cliffords through the tableau before they can be included. One such method is to represent the Clifford gates as Pauli exponentials. If we can then move these through the tableau, then can we synthesise the resulting Clifford Pauli exponentials and include the gates into the tableau from the correct side. First, we define Pauli exponentials for the generators.

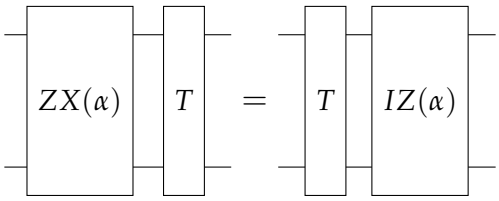
Lemma 35 (Cliffords as Pauli Exponentials): *The following equations hold.*

$$\begin{aligned} S_i &\approx \overrightarrow{Z_i}\left(\frac{\pi}{2}\right) \\ H_i &\approx \overrightarrow{Z_i}\left(\frac{\pi}{2}\right)\overrightarrow{X_i}\left(\frac{\pi}{2}\right)\overrightarrow{Z_i}\left(\frac{\pi}{2}\right) \\ CX_{i,j} &\approx \overrightarrow{Z_iX_j}\left(\frac{\pi}{2}\right)\overrightarrow{Z_i}\left(-\frac{\pi}{2}\right)\overrightarrow{X_j}\left(-\frac{\pi}{2}\right) \end{aligned}$$

Proof of Lemma: These can be easily checked numerically. ■

For pushing a Pauli exponential $\vec{P}(\alpha)$ through the tableau, we first make the simplifying assumption that $Q \in \{I, Z, X\}$ for $Q \in \vec{P}$. We do this as, for our context, Y will not be appearing in Pauli exponentials being pushed through a unitary tableau. Next, we multiply the output stabilisers for all input stabiliser generators in $\vec{P}$. In our example above, if we wished to push $\overrightarrow{ZX}$ through the unitary tableau, we multiply the output stabilisers corresponding to Z_1 and X_2,

e.g., $\overrightarrow{ZZ}$ and $\overrightarrow{ZI}$ to give $\overrightarrow{IZ}$. $\overrightarrow{IZ}$ is then the new string for the Pauli exponential. In other words, letting T be the unitary tableau above, the following holds.



To push a Pauli exponential through the unitary tableau, we incur a cost of $\mathcal{O}\left(q^2\right)$, since q Pauli strings of length q are multiplied together in the worst case. To synthesise a Clifford Pauli exponential creates $\mathcal{O}\left(q\right)$ gates from the Pauli exponential synthesis seen in Chapter 2. Each of these Cliffords taking $\mathcal{O}\left(q\right)$ to include in the tableau using Lemma 18, Lemma 19, and Lemma 20 over every generator. This gives a running time of $\mathcal{O}\left(q^2 \cdot n\right)$ for translation. In our case, all Pauli exponentials pushed through the tableau contain at most 2 active letters. Therefore actual cost of commuting Cliffords through the unitary tableau becomes $\mathcal{O}\left(q \cdot n\right)$. However, we still have a worst-case cost of $\mathcal{O}\left(q^2 \cdot n\right)$ for inclusion.

By initially placing this tableau at the end of the circuit, we can push the circuit through it collecting Cliffords until the start of the circuit is reached. At this point, since we are using the $|0\rangle^{\otimes q}$ state, our stabilisers are the outputs corresponding to $Z_1, \ldots Z_q$. We now define the following rules for including Clifford gates and pushing a Pauli exponential through a tableau.

4.3.3 Pauli DAG

The Pauli DAG needs to allow for the following operations.

- Adding a Pauli
- Removing a Pauli
- Merging two Paulis if they are compatible to be merged

From the Pauli DAG Merging Theorem, we need to ensure that the direct parent sets $\rho\left(\vec{P}\right)$ are maintained for every Pauli. Maintaining these $\rho\left(\vec{P}\right)$ sets allows us to make efficient and easy comparisons to calculate whether two Paulis exponentials can be merged in the DAG. We can track both the forward and edges in the DAG to efficiently track the sets during insertions and removals.

From the Pauli DAG's definition, edges only go from the Pauli exponentials appearing earlier to those later in the circuit. As a result, it is natural to enforce an insertion order from the last to the first. The insertion procedure then becomes simple, presented in the pseudocode below.

```
void InsertPauliExponential(P: PauliExponential) {
    ExistingPaulis.add(P)
    for(Q in ExistingPaulis) {
        if (P⃗Q⃗ ≠ Q⃗P⃗) {
            AddEdge(P, Q)
        }
    }
}
```

To remove a Pauli exponential from a graph, we must ensure that we remove all edges incident to the Pauli exponential and that the ρ relation is maintained. Since we define our edge set by commutation, removing a Pauli exponential does not affect any commutation for the remainder of the graph. Not changing commutation simplifies the Pauli exponential removal algorithm while maintaining our definition of ρ.

```
void RemovePauliExponential(P: PauliExponential) {
    ExistingPaulis.remove(P)
    for (e in BackEdges(P)) {
        RemoveEdge(e)
    }
    for (e in ForwardEdges(P)) {
        RemoveEdge(e)
    }
}
```

Finally, the two conditions of the Pauli DAG Merging Theorem for merging Pauli exponentials are that their product is a state stabiliser and that their parent sets ρ in the DAG are equal. We already know the ρ sets for every Pauli exponential in the graph, so it remains to maintain them after such a merge. Since a Pauli exponential is being removed and the other only changing phase, the graph's connectivity does not change. The constant connectivity simplifies the merging algorithm to the following.

```
void MergePauliExponentials(P: PauliExponential, Q: PauliExponential,
                            T: StabiliserTableau) {
    if (!T.CanCreate(P⃗Q⃗)) {
        return
    }
    if (BackEdges(P) ≠ BackEdges(Q)) {
        return
    }
    P.CombineWith(Q, T)
    RemovePauliExponential(Q)
}
```

Since the product $\vec{P}\vec{Q}$ may produce a negative sign, we must pass the tableau to the combination procedure.

4.4 Pauli DAG Optimistaion

There are a few ways to optimise the Pauli DAG once it is created. The primary method will be through repeated applications of the Pauli DAG Merging Theorem. However, we can observe that when merging two phases, we may end up with a Clifford. This Clifford can then be pulled to the start of the circuit and included in the stabiliser-destabiliser tableau. Finally, it may be possible to apply Lemma 22 to remove a Pauli exponential entirely.

4.4.1 Merging Pauli Exponentials

Applications of the Pauli DAG Merging Theorem can allow the merging of Pauli exponentials in the DAG. We also note that by Theorem 3, none of these merges will cause additional Paulis to be able to merge that could not before. Since no additional merges can occur, we can do a single pass of the Pauli DAG in any order. The only time further iterations will need to occur is when Cliffords are pulled out. By transitivity from Theorem 4, we also see that it does not matter which of the two the resulting Pauli exponentials remains. These facts allow us to merge greedily without worrying about disrupting later merges.

One such method is to check all pairs of Pauli exponentials. This method considers $\mathcal{O}\left(n^2\right)$ pairs where n is the number of Pauli exponentials. Another approach is to create a map from sets of ρ to Pauli exponentials. This method has the advantage of having efficient checks by taking advantage of the transitive properties of merging. However, the construction of such a map itself takes $\mathcal{O}\left(n^2\right)$ since the key is of size $\mathcal{O}\left(n\right)$, and n Pauli exponentials are to be inserted. Therefore, checking all pairs was chosen as the more reliable and simple option

for this design.

As far as is known, we cannot do better than this complexity in the worst case. However, we can improve substantially by recognising some properties of the ρ set. We must choose between checking all other nodes that appear before or after the considered node in the topological ordering. By checking forwards, we can never gain prior knowledge about later parent sets that would allow us to shortcut the checking. However, by checking backwards, we can note that we only need to check as far as it takes for a Pauli exponential that does not commute with our Pauli exponential to appear. Since we can merge only if the ρ sets are the same, by checking backwards, the first Pauli exponential that does not commute will be in the parent set of the considered Pauli exponential. However, all earlier will not contain this Pauli exponential in their parent set.

In the worst case, this optimisation does not change our overall complexity. To see this, we can consider the instance where all Pauli exponentials commute but no products of their Pauli strings stabilise the initial Clifford state. In this case, no Pauli exponentials can merge, but we also cannot shortcut our check. However, in real-world circuits, these conditions are unlikely, and it would likely be clear from their definition that this technique is not well suited for optimisation. As a result, this technique was chosen for the compiler's design.

A vital component of the merging conditions is that the product of the two Pauli strings of the exponentials is a stabiliser for the circuit. As shown in [1], the tableau presents the q stabiliser generators for the circuit, which we can use to perform this check. We must find a stabiliser subset such that their product equals that of the Pauli strings up to a negation. We can represent each Pauli string by a vector, as shown below. This widley used encoding of the Pauli

matrices is known as the binary symplectic representation in literature [1, 24, 30].

$$I = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad Z = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$
$$X = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad Y = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Then, we can recognise that the summation of any two of these vectors under the field $\mathbb{F}_2$ is equal to their product up to a constant factor. This fact reframes the problem as finding a linear combination under the field $\mathbb{F}_2$. This technique allows us to pack our stabilisers into a $2q \times q$ matrix and perform gaussian elimination.

To make this more apparent, we consider a 3-qubit Clifford state $|\psi\rangle$ with stabiliser generators $\overrightarrow{ZZI}$, $\overrightarrow{XXX}$, and $\overrightarrow{ZIZ}$. We wish to know if we can produce a stabiliser $\pm\overrightarrow{XYY}$. Recall that since we only care about changing the Pauli string in the Pauli exponential, the sign of the Pauli string does not affect this. We pack these stabiliser generators into a matrix as follows.

$$S = \begin{bmatrix} Z & X & Z \\ Z & X & I \\ I & X & Z \end{bmatrix}$$

Using the binary symplectic representation, this becomes.

$$S = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

We can write out the target in a similar form to get the following equation.

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

This is an underdetermined set of linear equations so is not gauranteed to have a solution, but performing Gaussian elimination under $\mathbb{F}_2$ results in the following solution.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

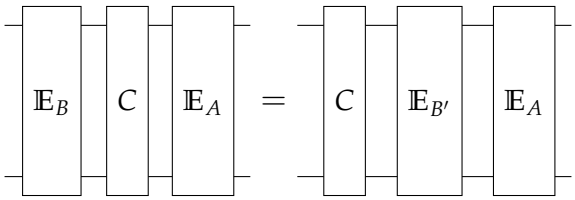
From here we can read off the solution as $x_1 = 1, x_2 = 1, x_3 = 1$. Verifying through multiplication, we see that $\overrightarrow{ZZI}\overrightarrow{XXX}\overrightarrow{ZIZ}$ equals $\overrightarrow{XYY}$ up to a negation.

Once the unknown vector is found, it remains to take the product of the stabilisers. This product still needs to be done as we only have the product up to a negation. Since they commute, their product will never contain an imaginary component. This method gives an $\mathcal{O}\left(q^3\right)$ algorithm to check whether we can apply the merge rules and identifying any sign updates that need to occur during the merge.

4.4.2 Pulling Out Cliffords

The Cliffords that may appear can be pulled to the front of the circuits by Theorem 1. However, since we are in the context of the Pauli DAG, we can do slightly better. Instead of choosing some topological ordering and then pushing the synthesis of each of the Cliffords through the circuit, we can recognise that we only need to move the Clifford past the ρ set of the Clifford. This fact holds because everything else in the circuit can either appear afterwards or will commute with the Clifford gate set.

Once Cliffords have been pulled out of the Pauli DAG, it splits the circuit into two sections, those appearing before the Clifford and those appearing after the Clifford labelled $\mathbb{E}_B$ and $\mathbb{E}_A$ respectively.



It is clear that connectivity does not change within $\mathbb{E}_A$. We can also see that the connectivity within $\mathbb{E}_B$ is the same as in $\mathbb{E}_{B'}$ from the following lemma.

Lemma 4.1. *Let $\vec{P}(\alpha)$, $\vec{Q}(\beta)$ be Pauli exponentials such that $\vec{P}\vec{Q} = \vec{Q}\vec{P}$. Letting C be a Clifford unitary, then $\left(C\vec{P}C^\dagger\right)\left(C\vec{Q}C^\dagger\right) = \left(C\vec{Q}C^\dagger\right)\left(C\vec{P}C^\dagger\right)$.*

Proof of Lemma:

$$\begin{aligned}\left(C\vec{P}C^\dagger\right)\left(C\vec{Q}C^\dagger\right) &= C\vec{P}\vec{Q}C^\dagger \\ &= C\vec{Q}\vec{P}C^\dagger \\ &= \left(C\vec{Q}C^\dagger\right)\left(C\vec{P}C^\dagger\right)\end{aligned}$$

■

However, the connectivity between the first and second groups can change due to the operation, as shown below.

$$\vec{X}(\alpha)\,\vec{Z}\left(-\frac{\pi}{2}\right)\vec{Y}(\beta) = \vec{X}(\alpha)\,S^{\dagger}\vec{Y}(\beta) = \vec{X}(\alpha)\,\vec{X}(\beta)\,S^{\dagger}$$

This connectivity change means we must reconstruct the edges of the graph to ensure our ρ sets are correct. It also changes the stabilisers that can be applied to the second group, potentially allowing additional merges. Since the stabilisers are conjugated in the same way as the Pauli exponentials in the first group, we can find no further merges within the first group.

4.4.3 Cancelling Pauli Exponentials

When cancelling a Pauli exponential, we can apply Lemma 22. Assuming we perform as many merges as possible up front, there will only ever be a single Pauli exponential in the circuit that we can cancel from Lemma 32. Using the simple properties of Lemma 31, we can remove any Pauli exponentials with a phase of 0 and perform any cancellations in a single pass through any topological ordering. Any Clifford in the parent set of a Pauli exponential with a phase of 0 is removed before we check if the Pauli exponential can be cancelled. Since Pauli exponentials with phase 0 equal the identity, we also do not need to recompute ρ sets.

4.5 Pauli Exponential Synthesis

Once optimised, we finally need to synthesise the Pauli exponentials that still exist in the circuit. A common technique is partitioning the Pauli exponentials into mutually commuting groups that still obey a topological ordering. Then the groups are converted into phase polynomials by conjugating with some Clifford circuit [8, 36]. Finally, the phase polynomials are synthesised using some phase polynomial synthesis techniques such as Gray-Synth [4]. In this section, we discuss these techniques in more depth.

4.5.1 Choosing Groupings

Since we need to maintain a topological ordering while finding mutually commuting groups, we can use a greedy technique of deconstructing the Pauli DAG. Initially, the Pauli DAG will have a set of k Pauli exponentials that have empty ρ sets. These empty sets imply the existence of a topological ordering in which they appear as the first k elements. Since all of their ρ sets are empty, all of them commute with each other by definition. As a result, we can choose this as the first grouping. We can then remove these Pauli exponentials from the Pauli DAG and repeat this process until the Pauli DAG is empty. We are left with sets of mutually commuting Pauli exponentials.

Greedy grouping techniques like these are quite common in optimisation techniques [8]. In this context, we have a strong justification for it. Had no mergings occurred, we claim that any two Pauli exponentials that would have merged in the Pauli DAG would appear in the same group after the grouping phase. This is easy to see by considering the ρ condition of the Pauli DAG Merging Theorem.

If they could merge, they must have equivalent direct parent sets, so they will both have empty ρ sets for the same grouping.

Consequently, we recognise that it does not affect the optimisation outcome of which of the two Pauli strings we choose to remain after a merge. In the grouping phase, we can exchange these Pauli strings with each other using the Product Rotation Lemma.

4.5.2 String Reductions

Once the groupings have been chosen, we can apply the Pauli Rotation Lemma in what is referred to as the string reduction method. This method aims to improve the metrics resulting from the synthesis and follows the following pseudocode.

```
void StringReductions(stabilisers: Stabiliser[],
                      groups: PauliExponential[]) {
    for (group in groups) {
        stabilisers := CommutingStabilisers(stabilisers, groups)
        ReduceStrings(group, stabilisers)
    }
}
```

The reduce strings function applies the Product Rotation Lemma using the commuting stabilisers to change the Pauli strings according to some metric. Since the next steps of diagonalisation and Gray-Synth do not rely directly on the complexity of Pauli strings, we use a heuristic scoring function as a metric. For this, we first need to define the concept of size. We let k be the number of non I letters in the Pauli string, and l be the number of X and Y letters in the Pauli string. By looking at the synthesis of a Pauli exponential on its own as a result of

Lemma 5, we can see that the Pauli exponential will generate $2k$ CXs, $2l$ Clifford conjugating gates, and a single RZ gate.

Considering our priority of reducing the two-qubit gate count metric over the single-qubit gate count metric, we define our scoring function first to reduce the number of non I letters. Then, we break ties by minimising the number of non Z letters. We can encode this as a scoring function as follows. Let q be the number of qubits, k' and l' be the value of k and l after the application of the Product Rotation Lemma,

$$score = k - k' + \frac{l - l' + q}{2q}$$

Unfortunately, this scoring function is still a heuristic since it does not consider how the diagonalisation procedure can find a small Clifford set to conjugate the group with or how the Gray-Synth algorithm chooses to synthesise the resulting phase polynomials. However, we can consider our metric of CX gate count and overall gate count in the case of a single Pauli exponential. Regarding the synthesis as a result of Lemma 5, maximising this score function corresponds to minimising our metrics (recalling that the CX gate count is considered a higher priority). This link provides a strong justification for the usefulness of this technique.

4.5.3 Diagonalisation

Diagonalisation takes in a set of Pauli exponentials and finds a Clifford such that conjugating the set of Pauli exponentials transforms the set into a phase polynomial. The compiler's design uses the algorithm Alexander Cowtan et al.

presented in [8]. We choose this algorithm as it takes extra steps to minimise the size of the resulting Clifford unitary as much as possible.

4.5.4 Gray-Synth

Given a phase polynomial, assuming there are shared qubits in the Pauli strings, the resulting circuit can cancel out shared CXs compared to the naïve synthesis as a result of Lemma 5. One well-known algorithm for this procedure is called Gray-Synth by Matthew Amy et al., introduced in [4]. Gray-Synth is a standard algorithm used in multiple other pieces of work, so it was the natural choice for the design of this project [8, 10, 35].

4.6 Clifford Synthesis

We can convert the stabiliser-destabiliser tableau into a circuit using a series of gaussian elimination steps. For details on this procedure, the reader is directed to [1] or [24]. Unfortunately, the resulting set of Cliffords is not minimal in many cases. For example, the empty circuit is re-synthesised as a series of H and S gates. Many Clifford optimisation techniques exist to solve a problem like this. However, since the project aims to investigate different areas where this technique is applicable, a variant of the merging technique was used.

4.6.1 Conversion to Pauli Exponentials

The first step for performing the optimisation technique is to convert the Clifford state into a series of Pauli exponentials. Since the tableau synthesis only

generates a series of S, H, and CX gates, we can apply Lemma 35. Other techniques can synthesise Clifford gates into a minimal set of Clifford Pauli exponentials [28]. However, this technique was not used due to the algorithm's complexity and the primary goal of seeing the effects of the new contribution.

The same merging optimisation procedure is now applicable with a few minor alterations. We perform this optimisation using a default tableau, recalling that this is where the i^{th} stabiliser is Z_i. Since all the Pauli exponentials in this constructed circuit are Clifford, pulling Clifford Pauli exponentials to the start of the circuit does not make sense. Instead, we only pull Cliffords containing a π phase into the tableau. We choose these Pauli exponentials because of the following lemma.

Lemma 36: *Given a Pauli exponential* $\vec{P}(\pi)$*, we have* $\vec{P}(\pi) \approx \vec{P}$

Proof of Lemma: Using the Pauli exponential expansion of Lemma 4, we get the following.

$$\begin{aligned}
\vec{P}(\pi) &= e^{-i\frac{\pi}{2}\vec{P}} \\
&= \cos -\frac{\pi}{2} I + i \sin -\frac{\pi}{2} \vec{P} \\
&= 0 \cdot I - i\vec{P} \\
&\approx \vec{P}
\end{aligned}$$

■

Lemma 7 then tells us that including a Pauli string in the tableau only changes the sign of rows in the tableau. This fact makes conversion back to a circuit from the tableau trivial, only requiring us to check the sign of each stabiliser to determine how to synthesise it.

We can also note from Lemma 36 combined with Lemma 8 that removing Pauli exponentials with a phase of π does not affect the commutation rules of the Pauli DAG, and only affects signs. Therefore, we do not need to reconstruct the entire Pauli DAG during removal.

4.6.2 Pauli Exponential Synthesis

When synthesising the Pauli DAG containing Cliffords, we first recognise that the only phases of the Pauli exponentials are $\pm\frac{\pi}{2}$ since 0 phases are removed, and π phases are extracted to the front of the circuit. We also recognise there only to be three types of Pauli exponential in the circuit.

$$\overrightarrow{Z_i}\left(\pm\frac{\pi}{2}\right) \qquad \overrightarrow{X_i}\left(\pm\frac{\pi}{2}\right) \qquad \overrightarrow{Z_iX_j}\left(\pm\frac{\pi}{2}\right)$$

By the definitions given in Lemma 35, and since merges do not introduce changes to Pauli strings, the initial phases remain. We can use these definitions to define the following rules for synthesising each Clifford Pauli exponential. Applying this to each Pauli exponential in the circuit gives us a smaller Clifford set to optimise.

Lemma 37: *The following equalities hold up to a global phase.*

$$\begin{aligned}
\overrightarrow{Z_i}\left(\frac{\pi}{2}\right) &\approx S_i & \overrightarrow{Z_i}\left(-\frac{\pi}{2}\right) &\approx S_i^\dagger \\
\overrightarrow{X_i}\left(\frac{\pi}{2}\right) &\approx SX_i & \overrightarrow{X_i}\left(-\frac{\pi}{2}\right) &\approx SX_i^\dagger \\
\overrightarrow{Z_iX_j}\left(\frac{\pi}{2}\right) &\approx CX_{i,j}SX_jS_i & \overrightarrow{Z_iX_j}\left(\frac{\pi}{2}\right) &\approx CX_{i,j}SX_j^\dagger S_i^\dagger
\end{aligned}$$

Proof of Lemma: These follow directly from Lemma 35 and Definition 2.9. ■

4.6.3 Single Qubit Optimisation

Once the Cliffords have been reduced, we still wish to perform any single qubit optimisations. By the Euler decomposition of rotations, any single qubit unitary can be converted into a series of Z-rotation, then X-rotation, then Z-rotation. I.e., $U = RZ(\alpha) RX(\beta) RZ(\gamma)$. Since we are outputting $U3$ gates in our final circuit, we can shift any gates not blocked by CX gates as far left as possible and compress the final result into a $U3$ gate. The rules for including an angle into a $U3$ gate are as follows.

Lemma 38: *The following equations hold.*

$$RZ(\delta) RZ(\alpha) RX(\beta) RZ(\gamma) = RZ(\alpha + \delta) RX(\beta) RZ(\gamma) \tag{4.1}$$

$$RX(\delta) RZ(\alpha) RX(0) RZ(\gamma) = RZ(0) RX(\delta) RZ(\gamma + \alpha) \tag{4.2}$$

$$RX(\delta) RZ(\alpha) RX(\pi) RZ(\gamma) = RZ(0) RX(\pi + \delta) RZ(\gamma - \alpha) \tag{4.3}$$

$$RX(0) RZ(\alpha) RX(\beta) RZ(\gamma) = RZ(\alpha) RX(\beta) RZ(\gamma) \tag{4.4}$$

$$RX(\pi) RZ(\alpha) RX(\beta) RZ(\gamma) = RZ(-\alpha) RX(\pi + \beta) RZ(\gamma) \tag{4.5}$$

And the following equation holds for $\delta = \pm\frac{\pi}{2}$

$$RX(\delta) RZ(\alpha) RX\left(\pm\frac{\pi}{2}\right) RZ(\gamma) = RZ(-\delta) RX\left(\alpha - \delta \mp \frac{\pi}{2}\right) RZ\left(\gamma \mp \frac{\pi}{2}\right) \tag{4.6}$$

Proof of Lemma: These can be easily checked numerically. ■

To demonstrate this method, we begin with the Euler decomposition of the H gate, i.e., $RZ\left(\frac{\pi}{2}\right) RX\left(\frac{\pi}{2}\right) RZ\left(\frac{\pi}{2}\right)$ then show how to include the following series

of gates into the following unitary.

$$RZ\left(\frac{\pi}{2}\right) RX\left(\frac{\pi}{2}\right) RX\left(\pi\right)$$

We will denote the generic single-qubit unitary $U = RZ\left(\alpha\right) RX\left(\beta\right) RZ\left(\gamma\right)$ as $U = (\alpha, \beta, \gamma)$. This gives the following:

$$\begin{aligned} RZ\left(\frac{\pi}{2}\right) RX\left(\frac{\pi}{2}\right) RX\left(\pi\right) \left(\frac{\pi}{2}, \frac{\pi}{2}, \frac{\pi}{2}\right) &\overset{(4.5)}{=} RZ\left(\frac{\pi}{2}\right) RX\left(\frac{\pi}{2}\right) \left(-\frac{\pi}{2}, -\frac{\pi}{2}, \frac{\pi}{2}\right) \\ &\overset{(4.6)}{=} RZ\left(\frac{\pi}{2}\right) \left(-\frac{\pi}{2}, -\frac{\pi}{2}, \pi\right) \\ &\overset{(4.1)}{=} \left(0, -\frac{\pi}{2}, \pi\right) \end{aligned}$$

Chapter 5

Implementation

This chapter details the implementation decisions of the project as well as any difficulties reached throughout development. The main elements of algorithms are held within the design chapter of this report, but we will discuss specific details here. The main goals of the compiler are to try and match the meaning of the original circuit where possible while mapping to the same form.

5.1 staq

C++ was chosen as the base language for implementing the compiler for a mixture of its high efficiency and support for existing functionality. One of the more complex algorithms in the process is the Gray-Synth algorithm. The staq library was in part developed by the original creators of the algorithm, giving an efficient implementation for use in the new compiler's pipeline.

Alongside Gray-Synth, another complex component is parsing into a standard

form. staq provides functionality to translate any input QASM file to consist only of $U3$ and CX. Translating the input to this form reduces the complexity of processing the input for the compiler rather than considering all gate types. An alternative would be to fix a smaller gate set. However, this technique allows the compiler to be used across any circuit. Since part of the optimisation procedure is converting to Pauli exponentials and a stabiliser tableau, changing the gate set to this form does not affect the compiler's output.

5.2 Parsing the QASM Files

staq offers a plethora of tools for working with QASM abstract syntax trees (ASTs). However, in this case, the main computation steps take place in custom data structures. As a result, a standard way of converting the input file into a local form must occur. staq allows all gate definitions to be inlined, leaving us with $U3$ and CX gates. From here, we wish to generate as few internal gates as possible. We can recognise a few forms such as $U3\left(\frac{\pi}{2}, 0, \pi\right)$ as representing Hadamard gates, and $U3\left(\theta, -\frac{\pi}{2}, \frac{\pi}{2}\right)$ for RX gates to simplify this process, but generally use the form $U3\left(\theta, \phi, \lambda\right) \approx RZ\left(\lambda - \frac{\pi}{2}\right) RX\left(\theta\right) RZ\left(\phi + \frac{\pi}{2}\right)$. This method allows us to extract a series of RZ, RX, and CX gates.

Given a gate set, we recognise all RX and RZ gates with non-Clifford phases. These gates are represented as single-qubit Pauli exponentials that we can use in the algorithms for building the data structures. All other gates are Clifford gates, so we store them as just that.

5.3 Qubits Management

Since we wish to have the output format have a similar meaning to the original circuit, it is essential to allocate identical qubits with the same names in the correct ordering. To do this, we can implement a qubit manager that stores the original information about the qubit allocations. When a qubit reference from the QASM file appears, this component provides an integer representation for the qubit to reduce memory usage.

5.4 Conversion to Tableau-Pauli-DAG Form

As discussed in the design chapter, there are two techniques for converting to this form: pushing a full tableau back through the circuit vs pushing individual Cliffords back. The complexity of pushing the entire tableau through is $\mathcal{O}\left(q^2 \cdot n\right)$ compared to $\mathcal{O}\left(n^2 + q \cdot n\right)$. In most cases, $q \ll n$, so $\mathcal{O}\left(q^2 \cdot n\right)$ is cheaper than $\mathcal{O}\left(n^2 + q \cdot n\right)$. However, where $q^2 > n$, we see that $\mathcal{O}\left(q^2 \cdot n\right)$ dominates $\mathcal{O}\left(n^2 + q \cdot n\right)$. In other words, for smaller circuits with large tableaus, more specifically where $q^2 > n$, the cost of pushing through a tableau is far greater, so the simple Clifford pushing technique should be used.

We can take this further by recognising that we can push the first $\mathcal{O}\left(q^2\right)$ Cliffords through Pauli exponentials as, with lower complexity and constants, the technique is more efficient. Including a single Clifford into the unitary tableau being pushed through requires a decomposition into Pauli exponentials. Then, the Pauli exponentials must be commuted with the tableau and synthesised back into Cliffords. Finally, we must include the gates in the tableau.

Once these first $\mathcal{O}\left(q^2\right)$ Clifford gates are pushed through the circuit, a tableau can be placed just before them in the circuit. The Clifford gates can be pulled into the tableau freely without extra computation. From an implementation perspective, we track the first $\mathcal{O}\left(q^2\right)$ Clifford gates, iterating backwards through the list of gates. When coming across any Pauli exponential, we conjugate it with all the Clifford gates we have currently seen. Once more Clifford gates are found, we include all Clifford gates into a tableau and continue by pushing the tableau to the start of the circuit.

We can add Pauli exponentials to the Pauli DAG as soon as they are pushed through the tableau or are conjugated by all Clifford gates appearing after them. The construction algorithm set out in Section 4.3.3 noted that reverse construction is favourable, so we need not wait until all are processed before inclusion.

5.5 Merging Algorithm

The next step of the implementation is handling merges. Once we have constructed the Pauli DAG, the number of checks we need to perform is $\frac{n(n-1)}{2}$ for each merging iteration (each pair). In many cases, we can observe that many circuits have few merges that occur beyond the start of the circuit. As a result, a majority of the checks will not cause any change to the Pauli DAG. Therefore, we can perform all comparisons of parent sets to find merges and store merges to be completed at the end.

This observation gives a framework that can be trivially parallelised. We can define an ordering to the nodes we wish to consider before placing every node into a concurrent queue shared between workers in that order. Then, each worker takes a batch of Pauli exponentials to consider. The worker only con-

siders pairs with Pauli exponentials appearing later in the fixed ordering for each batch. We can choose any topological order for simplicity, providing a reasonably balanced workload between workers.

Once a worker identifies two Pauli exponentials that can merge, it stores any information required for that merge, including the sign of the stabiliser that causes the merge. That Pauli exponential is removed from later consideration as we will have merged it. We note that this does not cause us to miss any merges by the transitivity of merging. All information is stored thread locally, and once all workers have finished, the main process can apply all merges found. After performing each merge, we must remove the later node. Otherwise, removing the later node from consideration while finding pairs is not safe.

Unfortunately, this method does come with overhead if there is a Pauli exponential that can be merged with many others. Assuming the node merges with k elements, then by transitivity, if different workers process all k elements, each will find it can merge with later Pauli exponentials. This gives $\frac{k(k-1)}{2}$ impossible merges being stored. Thankfully, we miss a lot of these extra merges in practice by choosing our ordering as a topological ordering and recognising that most merges will be close to each other in the ordering.

5.6 Pauli DAG

A large memory consumer is the storage of both forward and back edges needed for maintaining ρ sets. However, this is not entirely true. During the merging phase, we do not change whether or not two Pauli exponentials can merge. That only occurs when Clifford gates are pulled out of the circuit. However, the entire DAG requires reconstruction when Clifford gates are pulled out of the circuit.

Therefore, we can choose not to store the forward edges and only remove the back edges to remove nodes once merges occur. This memory optimisation does cause an increase in the cost of comparisons between back edge sets for instances where there are many merges. Still, it cuts down on the number of edges created initially and the cost of removing nodes.

This memory optimisation changes how we find groups and perform cancellations. First, we can recognise that cancellations can only occur for nodes that do not have parents in the Pauli DAG. This group corresponds to the first set of groupings. Then, we can notice that since we still have back edges, we have a record of all commutation between Pauli exponentials. Finally, we need to recognise that cancellations must occur before a later node has its grouping decided. Otherwise, a Pauli exponential that should be in the first grouping may appear later as the first grouping contains a cancellable Pauli exponential that does not commute with it. Putting all of this together, we get the following procedure.

```
Groups Groupings() {
    Groups := [[]]
    T := tsort()
    for (t in T) {
        for (g in Groups.reverse()) {
            if (t commutes with all in g) {
                current = g
            } else {
                break
            }
        }
        if (current is the first group) {
            if (Cancel(t)) {
                continue
            }
        }
        if (current = []) {
            Groups.append([])
        }
        current.append(t)
    }
}
```

We can combine this memory-saving technique with a more memory-efficient data structure for unordered_set to save massively on memory. We use the sparsepp set for its high efficiency yet very low memory usage for the implementation [29].

5.7 String Reduction

For the string reduction procedure, it is unclear whether there is an efficient algorithm that, given an initial Pauli and a set of stabiliser generators to multiply it with, can generate the smallest resulting Pauli string. As a result, we can use a randomised greedy algorithm to apply stabilisers to the string and take it if it is better. We take random subsets of the stabiliser set for each Pauli exponential and use them if they improve the score function.

5.8 Key Issues With Implementation

The most challenging problem with the implementation was ensuring the correctness of the code. It is often challenging to verify that signs are being carried through correctly since the rules around signs are not trivial for Pauli matrix multiplication. Thankfully, these errors were easy to spot in the integration testing phase, where they could be ironed out.

5.9 Testing

Google Test was chosen as the library for unit testing as a reliable and well-engineered tool with industry usage [12]. All separate components of the system were unit tested using the Google Test, making finding errors in the implementation much more straightforward.

Integration tests were written alongside unit testing to check full system correctness. A subset of the benchmark data was chosen as a test verification set. A script was then used to automatically run the compiler on the code, where the quantum circuit library PyZX was used to compare the input and output QASM files up to a global phase [17].

Chapter 6

Results

This chapter details the benchmarking of the resulting compiler. First, the methodology and parameters taken into consideration while benchmarking are discussed. Then, the data will be presented and discussed.

6.1 Methodology

The project's goal is first to test the limits of this as a technique for optimisation. Then, test the feasibility of it as a method concerning running time. Finally, see how feasible this is across different types of circuits. As a result, we track the following metrics.

- The gate, CX, and non-Clifford count and depth before and after optimisation
- The gate and CX counts without string reductions
- The gate, CX, and non-Clifford depth without string reductions

- Number of trivial merges that took place
- Number of Clifford removals that took place
- Number of cancellations that took place
- The string reduction score
- Time taken

This set of metrics allows us to answer all of the above questions. We convert all gates in the input circuits to $U3$ and CX gates. This conversion combats issues where some gates generate more $U3$ gates than necessary to avoid global phases. Since we are not considering global phases in this context, we ignore these discrepancies. The input circuit is set to be as similar as possible to the original circuit. This restriction means that manual checks took place for the preprocessing of the circuits to process different gate types properly.

Unfortunately, some of the circuits included in the benchmark sets allowed more complex gates not supported by the data structures here. These include mid-circuit measurements and reset gates. These circuits were omitted from the benchmark sets used. Another simplifying assumption is that the input state is fully prepared as $|0\rangle^{\otimes q}$, and any end-of-circuit measurements are ignored. Ignoring the end of circuit measurements does not affect the procedure since it just deals with the simplification of the state. However, oracle circuits generally consist of many Toffoli gates that will reduce to a Clifford state when the $|0\rangle^{\otimes q}$ state is applied.

For this reason, the benchmark sets have been split into oracle and non-oracle datasets. The full results for all benchmark sets are provided as a spreadsheet in the implementation's source code for the interested reader. They are omitted

here for brevity.

6.2 Running Parameters and Device

The compilation program was run on an AMD Ryzen 9 5900X 12-Core CPU running at 4.2Ghz, combined with 32GB of Corsair Vengeance LPX DDR4 memory running at 3600 MT/s with an 18-22-22-42 CAS Latency, and a Sabrent Rocket 4.0 1TB NVMe SSD. The program was compiled and run on Ubuntu 22.04.1 LTS using g++ (Ubuntu 11.2.0-19ubuntu1) 11.2.0 with the following compiler arguments: -std=c++2a -Ofast -DNDEBUG. The program was run with 12 threads, utilised during the merging phase of the procedure. The build system used was Bazel, chosen for the author's familiarity with the software and ease of use for external libraries [11].

6.3 Benchmark Sets

The original use case for this method is within Chemistry circuits. We wish to see how applicable this method is for both circuits of this form and more general circuits. As a result, the general purpose QASM Bench benchmark set will be used to evaluate general circuits [20]. Secondly, the benchmark sets produced by TKET provide both Chemistry circuits and general circuits [6]. In combination, these two sets provide enough variety in size and complexity.

6.3.1 TKET Benchmarking

The TKET Benchmarking dataset includes both chemistry sets formed from the UCCSD ansatz method with a single Trotterization step and regular circuits. The chemistry sets are split by molecules and have three components. A complete representation (cmplt) or one with frozen orbitals (frz), using Jordan-Wigner (JW), Bravyi-Kitaev (BK), or Parity (P) qubit encodings, and the size of the basis set used either sto3g, 631g, ccpvdz, or ccpvtz. These circuits range anywhere from 4 qubits up to 56 and from 100 gates to 1.25 million. The QASM files used for the benchmarks contain different molecules, including H2, H4, H8, LiH, NH, H2O, CH2 and C2H4.

A large portion of the regular circuits in the TKET Benchmarking set are oracle circuits. There are still many circuits that are not oracle circuits that will be of further interest. The sizes of the circuits are from 20 gates up to around 10,000.

6.3.2 QASM Bench

The QASM Bench benchmark set provides circuits of three different sizes: small, medium, and large. The small circuits generally have at most 10 qubits and only a few hundred gates. The medium circuits have up to 20 qubits and up to a few thousand gates. The large set has up to 500 qubits and anywhere from a few thousand to millions of gates. The circuits are often hand optimised or come from machine learning contexts. As a result, we are unlikely to see merges beyond trivial merges.

6.4 TKET Benchmarking Results

This section is split by the types of circuits being processed. This splitting gives a more precise context to where this technique is more applicable by splitting in this way and, as such, is better for a fair analysis.

6.4.1 Chemistry Circuits

Starting with the Chemistry circuits, no trivial merges occurred in any circuits. Unfortunately, the number of non-trivial merges remained very small, save a few outliers. For H2 complete circuits in the sto3g basis, 8 of the 12 original Pauli exponentials were merged for all qubit encodings. For H4 complete circuits in the sto3g basis, 10 of the 160 Pauli exponentials were merged. Aside from these cases, the number of merges occurred was 2 for every other chemistry circuit. This low merge count is thought to come from the order of Pauli exponentials chosen for the QASM files. Since these circuits come from Hamiltonians, we can choose any ordering of the Pauli exponentials when we create the circuit. Therefore, if there is initially an unfavourable ordering that does not allow for merges to occur, the resulting circuit will not permit merges. This unfavourable ordering can manifest in stabilisers being blocked from propagating far into the circuits and transitive edges appearing between Pauli exponentials that could otherwise merge.

Another possible reason is that a similar technique may be applied to the Hamiltonians before they are synthesised into QASM files. This application would massively stunt the effectiveness of the process being applied afterwards, possibly leading to the results observed.

	Mean Reduction (%)		Max Reduction (%)		Min Reduction (%)	
Encoding	Count	Depth	Count	Depth	Count	Depth
JW	79.9	81.0	85.3	85.6	72.4	74.6
BK	71.9	74.0	81.6	81.8	61.9	63.7
P	70.3	72.6	83.3	83.7	59.4	61.6

Table 6.1: Comparing CX count and depth reductions for different qubit encodings for the full optimisation

Interestingly, the use of string reduction is highly effective against these circuits. Circuits saw up to 42% CX count and depth reductions and 36% total gate count and depth reductions from the repeated applications of string reduction techniques. Due to string reductions, the non-Clifford depths were slightly improved across nearly all circuits. A possible reason for this is that the Pauli exponentials that make up the circuits could allow a subset of the stabilisers to move through the entire circuit. This movement gives a large amount of freedom to string reduction techniques giving significant gains across every grouping found during synthesis.

The running time of the compiler on each of the Chemistry sets remains relatively low across the board. For even the largest dataset containing 1.25 million gates initially, the total running time is at most 40 seconds. For larger circuits, the majority of the running time for the program came from the construction of the Pauli DAG. The compiler stayed well within memory limits across all runs.

It can be seen in Table 6.1 that the JW encoding is much better suited for reduction by this technique showing a much greater reduction in both CX count and depth, while the BK and P encodings are very similar in their performance. However, in all cases, the performance seen is excellent. We see a slight improvement in the CX depth on top of the CX reduction suggesting improvements in the structure of the final circuit and gate reductions.

	Mean Reduction (%)		Max Reduction (%)		Min Reduction (%)	
Encoding	Count	Depth	Count	Depth	Count	Depth
JW	22.6	23.3	42.6	41.6	10.3	11.1
BK	6.0	6.9	22.6	23.1	1.2	1.3
P	12.4	12.1	29.5	29.8	5.1	5.2

Table 6.2: Comparing CX count and depth reductions for different qubit encodings gained by string reduction

Qubit Encoding	Mean Reduction Score (%)	Mean Reduction (%)
JW	5.1	22.6
BK	1.1	6.0
P	0.8	12.4

Table 6.3: Comparing CX count reduction for different qubit encodings gained by string reduction with the score function

Table 6.2 shows significant additional CX count and depth reductions by including the string reduction technique in the procedure. Once again, this impact is felt most when looking at circuits using the JW qubit encoding and less so by BK and P. The technique also seems better suited for the P qubit encoding than the BK. These reductions in CX count and depth suggest that the JW encoding has more redundancy than the other encodings.

These reductions allow us to look for correlations between the scoring function used for performing the string reductions and the actual reductions achieved. Instead of the raw score, we consider the average score per Pauli exponential. We can see from Table 6.3 that the more significant improvements correlate with a larger overall reduction score. However, the disparity between the average score and the relative improvements of BK and P suggests that the heuristic may be inaccurate. This disparity could be caused by strings seeing reductions that would have been later subsumed by the diagonalisation or phase polynomial synthesis steps. As the score function does not attempt to cross this complex optimisation landscape, it is expected not to see gains everywhere it predicts.

Qubit Encoding	CX vs Total Gate Count (%)
JW	74.9
BK	54.7
P	50.6

Table 6.4: Comparing CX count to total gate count for different qubit encodings before optimisation

	Mean Reduction (%)		Max Reduction (%)		Min Reduction (%)	
Encoding	Count	Depth	Count	Depth	Count	Depth
JW	84.7	78.0	90.7	84.2	79.2	70.1
BK	79.1	71.7	84.8	79.5	75.5	63.3
P	79.6	71.2	85.6	82.4	75.2	61.1

Table 6.5: Comparing gate count and depth reductions for different qubit encodings for the full optimisation

Another metric we consider is the overall reduction of gates in the circuit, not just CX gates. An important consideration is the ratio between the original gate count and the number of CX gates. As seen in Table 6.4, we can see that the CX count relative to the total gate count in the original circuit is much higher for the JW encodings, while the BK and P are very similar. These higher original counts could also have been the cause of the higher reductions seen earlier.

Looking at the gate reductions, we can see in Table 6.5 that the gate count reductions achieved are much more similar across the different encodings than the CX reduction. These results suggest that the BK and P qubit encodings contain more unnecessary single qubit unitaries in their computation. This observation makes sense by looking at the ratio of CX count to gate count. We see that these circuits' gate depth reduction is considerably lower than the gate count reduction in all cases. This provides evidence to suggest that while we are improving the number of gates in the circuit, we are losing some relative parallelism. Different synthesis techniques for mutually commuting groups could assist with such a problem.

	Mean Reduction (%)		Max Reduction (%)		Min Reduction (%)	
Encoding	Count	Depth	Count	Depth	Count	Depth
JW	17.1	17.8	36.9	36.1	7.1	7.6
BK	3.9	5.4	15.6	17.3	0.7	1.2
P	8.3	9.7	19.0	21.0	3.3	4.5

Table 6.6: Comparing gate count and depth reductions for different qubit encodings gained by string reduction

Qubit Encoding	Mean Reduction (%)	Max Reduction (%)	Min Reduction (%)
JW	0.3	3.5	-0.8
BK	7.1	12.0	0.5
P	8.7	13.9	0.9

Table 6.7: Comparing non-Clifford depth reduction for different qubit encodings for the full optimisation

Table 6.6 shows the (%) improvements for the single qubits are much less than for CX gates. Looking at the scoring function, this makes sense as the priority of the scoring function is to minimise the number of CX gates. We see similar performance between the different methods. We do see that the depth reduction is very similar to the reduction in the gate count, suggesting that this optimisation does not negatively impact the parallelism in the circuit.

Finally, we wish to look at the non-Clifford depth improvements across the different qubit encodings. We first examine the overall reductions in non-Clifford depth from the complete optimisation procedure.

As is shown in Table 6.7, there are minimal reductions in the non-Clifford depth with some cases showing an *increase* in non-Clifford depth. Unlike the other metrics, the data shows that circuits using the BK and P encodings enjoy much greater benefits from this optimisation than the JW encoding. A possible reason for these small improvements comes from a reduction in CX gates in the circuit. These allow more non-Clifford gates to be composed in parallel. The size of these reductions being so small and sometimes negative likely comes partly

Qubit Encoding	Mean Reduction (%)	Max Reduction (%)	Min Reduction (%)
JW	0.2	0.9	0.0
BK	1.6	4.6	0.04
P	0.4	2.8	-0.02

Table 6.8: Comparing non-Clifford depth reduction for different qubit encodings gained by string reduction

from the Gray-Synth algorithm not accounting for gate depth [4]. Applications of a different synthesis method for mutually commuting sets may yield better non-Clifford depth statistics. The larger improvements seen in the BK and P encodings suggest some hidden structure that is untangled during this synthesis allowing for further parallelism.

Considering non-Clifford depth from string reductions, Table 6.8 shows minimal benefits to this technique, but it is also unlikely to impact the non-Clifford depth metric negatively. The usage of diagonalisation and Gray-Synth after the string reduction technique makes it very difficult to predict how the final synthesis will appear. The non-Clifford count is hard to affect through changing Pauli strings and is a property not encoded into the scoring function.

6.4.2 General Circuits – Oracle

The system performed exceptionally well for all oracle circuits in the TKET benchmarking set, reducing the number of non-Clifford gates to 0 in all but one case. This reduction always came as a mixture of trivial merges, regular merges, and cancellations. In some cases, there were also significant amounts of Cliffords pulled out of the circuit leading to further merges and cancellations. The running time for all oracle circuits was less than a second. This spread of reductions shows that all techniques are helpful.

The only exception to reducing to 0 non-Clifford gates came from square_root_7. In this circuit, we saw a mixture of trivial and non-trivial merges and a single cancellation[1]leading to a 79% reduction in non-Clifford gates. Furthermore, string reduction methods were able to reduce the number and depth of CXs in the resulting circuit by 34% and the overall gate count and depth by 24%. The non-Clifford depth reduction from the complete optimisation was reduced by 62% with negligible benefits from string reductions. A possible reason for this outlier could come from some non-Clifford identity in the circuit that allows commutations beyond those allowed by the rules of commuting Pauli strings. A further investigation into this circuit could provide further insights into improving the technique described here.

6.4.3 General Circuits – Non-Oracle

For the general, non-oracle circuits in this benchmark set, all merges were trivial merges. This merge type led to a 57% reduction in the number and a 52% reduction in the depth of non-Clifford gates. Looking at the gate count difference due to the full optimisation, the number of CX gates in the circuit increased by at least 100%. Since all merges were trivial, phase-folding would have found these merges without increasing the gate count in the circuit. This increase likely comes from the re-synthesis of the Pauli exponentials into the final circuit and suggests this type of circuit is not well suited for this type of optimisation. Despite the high string reduction score for a few circuits, string reductions did not affect the resulting gate count across all circuits.

[1]Recall that the number of cancellations is at most 1 unless there are Clifford removals changing commutation

Metric	Small	Medium	Large
Mean Non-Clifford Reduction (%)	25.5	57.1	27.3
Mean Non-Clifford Depth Reduction (%)	14.2	40.4	-1444.9
Mean Trivial Merge (%)	29.0	91.9	83.3
Mean CX Reduction (%)	-22.3	-13.7	-2315.3

Table 6.9: Metrics for QASM Bench non-oracle circuits of all sizes

6.5 QASM Bench Results

Overall, the system performed very well on all oracle circuits within the QASM Bench dataset, reducing the non-Clifford count to 0 in all cases. Across the board, average non-Clifford reductions for non-oracle circuits mainly come from trivial merges, as shown in Table 6.9[2]. However, many of these merges come from non-trivial applications of the Pauli Merge Rule. We also see from the data that the application of this technique heavily impacts parallelism. Even where the number of non-Cliffords is reduced, the relative reduction in the non-Clifford depth is relatively poor, with some cases giving substantial increases in the non-Clifford depth of the circuit.

Again from Table 6.9, we see CX count increases by applications of this technique. The specific circuits of vqe_uccsd_n4, vqe_uccsd_n6, and vqe_uccsd_n8 are somewhat exceptions to this. For these circuits, we see much better improvements of up to a 50% reduction non-Clifford count reduction and an 80% CX count and depth reduction. These results are more consistent with the results seen in the TKET Benchmarking dataset, suggesting that this technique is bested suited for application on circuits obtained from the UCCSD ansatz.

Considering string reductions for the circuits vqe_uccsd_n4, vqe_uccsd_n6 and

[2]The large 500 qubit circuit partially skews the data in this table for the large circuits. For full results, the reader is directed to the entire dataset.

Circuit	Total Gates	Non-Clifford Count	Time (seconds)
ising_n26	100	280	0.13
vqe_n24	85,560	2,306,072	985.85
wstate_n27	52	157	0.109
dnn_n16	992	2,016	0.49
swap_test_n25	108	230	0.12
ising_model_n500	1,996	5,494	131.649
multiplier_n25	770	1,743	0.27
bigadder_n18	112	284	0.4
qft_n20	570	970	0.14

Table 6.10: Running times for QASM Bench large circuits

vqe_uccsd_n8, we see CX count reductions of over 22% and over 14% total gate reduction. We also see a 31–35% CX depth reduction, 30–40% overall gate depth reduction, and 20–36% overall non-Clifford depth reduction. All other circuits do not change from string reductions.

The time and memory utilisation of the small and medium datasets were minimal, taking at most 1 second. For the large datasets, Table 6.10, we see that the running times for each circuit are reasonable and scale well with the original circuit's size. In the case of vqe_n24, before a memory-efficient hash set was used, this circuit ran out of memory. However, after the change, the memory usage remained at most 50%. These factors suggest that these methods are feasible for application in current compilers, adding very little overhead.

Chapter 7

Conclusions

This project has introduced a new theory surrounding the merging of Pauli exponentials within circuits composed with a Clifford state. We use existing techniques to translate general circuits to this form, allowing the application of the technique to general circuits. These circuits are then synthesised using existing techniques of diagonalisation and phase polynomial synthesis combined with the new string reduction technique. This project provides an implementation of the techniques discussed that shows that they are efficient for usage in practice, even for circuits too large for current quantum computers. This efficiency comes from the Pauli DAG Merging Theorem and its surrounding theory that provides a small and efficient set of checks to apply all possible merges under a set of assumptions. This theorem subsumes the previous techniques of phase teleportation, phase-folding, and trivial merges that appear as special cases of the rule. This optimisation proved very efficient for certain circuit classes and significantly reduced tracked metrics.

This technique is very good at optimisation for oracle circuits, reducing the number of non-Clifford circuits to 0 in nearly all cases. This behaviour is expected as

the $|0\rangle^{\otimes q}$ states often reduce oracle circuits to some Clifford state. Unfortunately, this means it is unrealistic to see in practice. However, this also has applications in circuits where a partially non-Clifford initial state is used, making the classical simulation computationally intractable for larger circuits, but it would still permit the application of this technique. Aside from oracle circuits, the main benefits seen were in VQE circuits, especially those using the UCCSD Ansatz. For these circuits, we see a substantial reduction in CX count and depth from the original optimisation of up to 85%, seeing considerable, further benefits in CX count and depth of up to 42% from the string reduction technique.

The string reduction appears to be an excellent technique for reducing the number of Cliffords in the resulting circuit. In nearly all cases, string reduction either improved or gave no change to metrics depending on the circuit; in only two of the 266 test cases, we see an increase in non-Clifford depth by at most 1. All other metrics were improved or stayed the same by this heuristic. The running time of the optimisation stays within reasonable limits for all circuits making this technique feasible in practice.

7.1 Future Work

The section details recommendations for potential future work to continue this project. The proposed work involves in-place merging, improvements of techniques for implementation, better heuristics for the synthesis stage of the optimisation, and the expansion of the implementation to support other gate types for broader applications and study of the technique.

7.1.1 In Place Merging

A considerable drawback of this technique for optimisation is the requirement to translate to Clifford-Pauli-Exponential form to gain access to the stabilisers and Pauli DAG. Since many circuits are hand optimised to reduce their Clifford count, in many cases, the re-synthesis of this form introduces far more gates than in the original circuit. Future work should include a formalism for performing these merges that captures this relationship and allows these merges to occur without changing the structure of the circuit. This technique is used by algorithms like phase-folding and phase teleportation and has been shown to be successful [2, 16]. Not changing the structure of the original circuit would make the technique more applicable to general circuits.

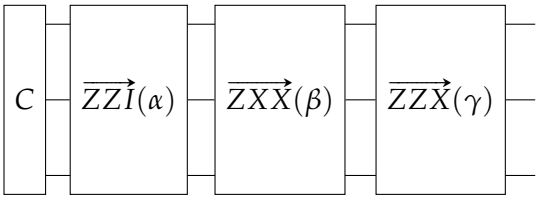
7.1.2 Memory Efficient Merges

The first area for optimisation is to find more memory-efficient ways of computing the conditions of the Pauli Merge Rule. While the Pauli DAG Merging Theorem does provide a good characterisation that can be efficiently implemented, for this technique to be efficient, it requires the full construction of the Pauli DAG. Some ways to improve this method could be to: recognise portions of the circuits that no stabilisers can reach by Lemma 34 or provide a new characterisation of the Pauli DAG Merging Theorem that does not require the full construction of the Pauli DAG.

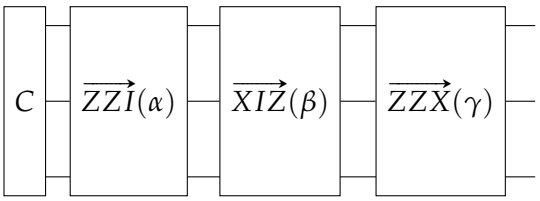
7.1.3 Better Groupings

The current method for choosing the groupings for synthesis, while well founded in theory, does not utilise the Product Rotation Rule to change the groupings. Circuits can exist where the application of the Product Rotation Rule allows for larger groupings to be found or different groupings better suited for later synthesis stages. We show one example below where the application of the Product Rotation Rule allows for the inclusion of multiple Pauli exponentials into a group.

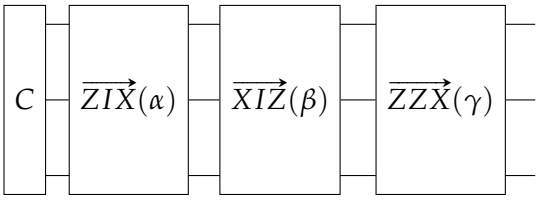
We consider a state where all consecutive Pauli exponentials do not commute (resulting in multiple groups). For example,



We can define C such that $\overrightarrow{YXY}$ and $\overrightarrow{IZX}$ are stabilisers of it. Then, using the stabiliser $\overrightarrow{YXY}$, we apply the Product Rotation Rule to $\overrightarrow{ZXX}$ resulting in the following state.



Finally, we apply the Product Rotation Rule to $\overrightarrow{ZZI}$ using the stabiliser $\overrightarrow{IZX}$ giving us,



Where all Pauli exponentials now commute and form a single group.

7.1.4 Improved Heuristics for Pauli String Reduction

Currently, the heuristics for Pauli string reduction do not account for the diagonalisation and phase polynomial synthesis techniques. This future work would aim to find heuristics for navigating this complex search space to improve the heuristics used for string reductions. The techniques used for this may need to consider specific implementations for diagonalisation and phase polynomial synthesis. However, ideally, they would be agnostic of the implementations.

7.1.5 Improved Gate Support

Currently, the implementation provided only considers gates that can be decomposed to $U3$ and CX gates. This simplification does not account for many circuits that use more complex gates, such as reset gates and mid-circuit measurements that alter the Clifford tableau throughout the circuit. Future work in this area should investigate how those gates can be included in the implementation and benchmark the results of the technique for circuits of this form.

Bibliography

[1] Aaronson, Scott & Gottesman, Daniel. Improved simulation of stabilizer circuits. *Phys. Rev. A*, 70:052328, Nov 2004. doi: 10.1103/PhysRevA.70.052328. URL: https://link.aps.org/doi/10.1103/PhysRevA.70.052328.

[2] Amy, Matthew. *Formal Methods in Quantum Circuit Design*. PhD thesis, University of Waterloo, 2019. URL: http://hdl.handle.net/10012/14480.

[3] Amy, Matthew & Gheorghiu, Vlad. staq – A full-stack quantum processing toolkit. *Quantum Science and Technology*, 5(3):034016, jun 2020. doi: 10.1088/2058-9565/ab9359. URL: https://doi.org/10.1088/2058-9565/ab9359.

[4] Amy, Matthew & Azimzadeh, Parsiad & Mosca, Michele. On the controlled-NOT complexity of controlled-NOT–phase circuits. *Quantum Science and Technology*, 4(1):015002, sep 2018. doi: 10.1088/2058-9565/aad8ca. URL: https://doi.org/10.1088/2058-9565/aad8ca.

[5] Bravyi, Sergey & Gambetta, Jay M. & Mezzacapo, Antonio & Temme, Kristan. Tapering off qubits to simulate fermionic hamiltonians, 2017. URL: https://arxiv.org/abs/1701.08213.

[6] Cambridge Quantum. tket_benchmarking, 2020. URL: https://github.com/CQCL/tket_benchmarking.

[7] Cowtan, Alexander & Dilkes, Silas & Duncan, Ross & Simmons, Will & Sivarajah, Seyon. Phase Gadget Synthesis for Shallow Circuits. *Proceedings 16th International Conference on Quantum Physics and Logic*, jun 2019. doi: 10.4204/eptcs.318.13. URL: https://doi.org/10.4204%2Feptcs.318.13.

[8] Cowtan, Alexander & Simmons, Will & Duncan, Ross. A Generic Compilation Strategy for the Unitary Coupled Cluster Ansatz, 2020. URL: https://arxiv.org/abs/2007.10515.

[9] Cross, Andrew W. & Bishop, Lev S. & Smolin, John A. & Gambetta, Jay M. Open Quantum Assembly Language, 2017. URL: https://arxiv.org/abs/1707.03429.

[10] de Griend, Arianne Meijer-van & Duncan, Ross. Architecture-aware synthesis of phase polynomials for NISQ devices, 2020. URL: https://arxiv.org/abs/2004.06052.

[11] Google. Bazel, 2022. URL: https://www.bazel.build.

[12] Google. Google Test, 2022. URL: https://github.com/google/googletest.

[13] Gosset, David & Kliuchnikov, Vadym & Mosca, Michele & Russo, Vincent. An algorithm for the T-count, 2013. URL: https://arxiv.org/abs/1308.4134.

[14] Gottesman, Daniel. The Heisenberg Representation of Quantum Computers. *22nd International Colloquium on Group Theoretical Methods in Physics*, pages 32–43, 1999. doi: 10.48550/arXiv.quant-ph/9807006. URL: https://arxiv.org/abs/quant-ph/9807006.

[15] Grimsley, Harper R. & Claudino, Daniel & Economou, Sophia E. & Barnes, Edwin & Mayhall, Nicholas J. Is the Trotterized UCCSD Ansatz Chemically Well-Defined? *Journal of Chemical Theory and Computation*, 16(1): 1–6, 2020. doi: 10.1021/acs.jctc.9b01083. URL: https://doi.org/10.1021/acs.jctc.9b01083. PMID: 31841333.

[16] Kissinger, Aleks & van de Wetering, John. Reducing the number of non-clifford gates in quantum circuits. *Phys. Rev. A*, 102:022406, Aug 2020. doi: 10.1103/PhysRevA.102.022406. URL: https://link.aps.org/doi/10.1103/PhysRevA.102.022406.

[17] Kissinger, Aleks & Wetering, John van de. PyZX: Large Scale Automated Diagrammatic Reasoning. *Proceedings 16th International Conference on Quantum Physics and Logic*, 318:229–241, 2020. doi: 10.4204/eptcs.318.14. URL: https://doi.org/10.4204%2Feptcs.318.14.

[18] Knill, Emanuel & Laflamme, Raymond & Zurek, Wojciech H. Resilient Quantum Computation. *Science*, 279(5349):342–345, 1998. doi: 10.1126/science.279.5349.342. URL: https://www.science.org/doi/abs/10.1126/science.279.5349.342.

[19] Lee, Soonchil & Lee, Seong-Joo & Kim, Taegon & Lee, Jae-Seung & Biamonte, Jacob & Perkowski, Marek. The cost of quantum gate primitives. *Journal of Multiple-Valued Logic and Soft Computing*, 12:561–573, 01 2006.

[20] Li, Ang & Stein, Samuel & Krishnamoorthy, Sriram & Ang, James. Qasmbench: A low-level qasm benchmark suite for nisq evaluation and simulation, 2020. URL: https://arxiv.org/abs/2005.13018.

[21] Litinski, Daniel. A Game of Surface Codes: Large-Scale Quantum Computing with Lattice Surgery. *Quantum*, 3:128, mar 2019.

[22] Litinski, Daniel. Magic State Distillation: Not as Costly as You Think. *Quantum*, 3:205, dec 2019. doi: 10.22331/q-2019-12-02-205. URL: https://doi.org/10.22331/q-2019-12-02-205.

[23] Marinescu, Dan C. & Marinescu, Gabriela M. *Classical and Quantum Information*. Academic Press, Boston, 2012. ISBN 978-0-12-383874-2. doi: https://doi.org/10.1016/B978-0-12-383874-2.00002-3. URL: https://www.sciencedirect.com/science/article/pii/B9780123838742000023.

[24] Maslov, Dmitri & Roetteler, Martin. Shorter Stabilizer Circuits via Bruhat Decomposition and Quantum Circuit Transformations. *IEEE Transactions on Information Theory*, 64(7):4729–4738, 2018. doi: 10.1109/TIT.2018.2825602.

[25] McClean, Jarrod R. & Sung, Kevin J. & Kivlichan, Ian D. & Cao, Yudong & Dai, Chengyu & Fried, E. Schuyler & Gidney, Craig & Gimby, Brendan & Gokhale, Pranav & Häner, Thomas & Hardikar, Tarini & Havlíček, Vojtěch & Higgott, Oscar & Huang, Cupjin & Izaac, Josh & Jiang, Zhang & Liu, Xinle & McArdle, Sam & Neeley, Matthew & O'Brien, Thomas & O'Gorman, Bryan & Ozfidan, Isil & Radin, Maxwell D. & Romero, Jhonathan & Rubin, Nicholas & Sawaya, Nicolas P. D. & Setia, Kanav & Sim, Sukin & Steiger, Damian S. & Steudtner, Mark & Sun, Qiming & Sun, Wei & Wang, Daochen & Zhang, Fang & Babbush, Ryan. OpenFermion: The Electronic Structure Package for Quantum Computers. *Quantum Science and Technology*, 5(3):034014, jun 2020. doi: 10.1088/2058-9565/ab8ebc. URL: https://doi.org/10.1088/2058-9565/ab8ebc.

[26] Nam, Yunseong & Ross, Neil J. & Su, Yuan & Childs, Andrew M. & Maslov, Dmitri. Automated optimization of large quantum circuits with continuous parameters. *npj Quantum Information*, 4(1), may 2018. doi: 10.1038/s41534-018-0072-4. URL: https://doi.org/10.1038%2Fs41534-018-0072-4.

[27] Olson, Jonathan & Cao, Yudong & Romero, Jonathan & Johnson, Peter & Dallaire-Demers, Pierre-Luc & Sawaya, Nicolas & Narang, Prineha & Kivlichan, Ian & Wasielewski, Michael & Aspuru-Guzik, Alán. Quantum Information and Computation for Chemistry, 2017. URL: https://arxiv.org/abs/1706.05413.

[28] Pllaha, Tefjol & Volanto, Kalle & Tirkkonen, Olav. Decomposition of Clifford Gates, 2021. URL: https://arxiv.org/abs/2102.11380.

[29] Popovitch, Gregory. Sparsepp: A fast, memory efficient hash map for C++, 2018. URL: https://github.com/greg7mdp/sparsepp.

[30] Rengaswamy, Narayanan & Calderbank, Robert & Pfister, Henry D. & Kadhe, Swanand. Synthesis of Logical Clifford Operators via Symplectic Geometry. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 791–795, 2018. doi: 10.1109/ISIT.2018.8437652.

[31] Shor, Peter W. Scheme for reducing decoherence in quantum computer memory. *Phys. Rev. A*, 52:R2493–R2496, Oct 1995. doi: 10.1103/PhysRevA.52.R2493. URL: https://link.aps.org/doi/10.1103/PhysRevA.52.R2493.

[32] Simmons, Will. Relating Measurement Patterns to Circuits via Pauli Flow . *Proceedings 18th International Conference on Quantum Physics and Logic*, jun 2021. URL: https://doi.org/10.4204%2Feptcs.343.

[33] Soeken, Mathias & Meuli, Giulia & Schmitt, Bruno & Mozafari, Fereshte & Riener, Heinz & Micheli, Giovanni De. Boolean Satisfiability in Quantum Compilation. *Philosophical Transactions of the Royal Society A*, 378(2164), feb 2020. doi: 10.1098/rsta.2019.0161. URL: https://doi.org/10.1098/rsta.2019.0161.

[34] Steane, A. M. Error Correcting Codes in Quantum Theory. *Phys. Rev. Lett.*, 77:793–797, Jul 1996. doi: 10.1103/PhysRevLett.77.793. URL: https://link.aps.org/doi/10.1103/PhysRevLett.77.793.

[35] van de Griend, Arianne Meijer & Li, Sarah Meng. Dynamic qubit allocation and routing for constrained topologies by CNOT circuit re-synthesis, 2022. URL: https://arxiv.org/abs/2205.00724.

[36] van den Berg, Ewout & Temme, Kristan. Circuit optimization of Hamiltonian simulation by simultaneous diagonalization of Pauli clusters. *Quantum*, 4:322, 2020.

[37] Zhang, Fang & Chen, Jianxin. Optimizing T gates in Clifford+T circuit as $\pi/4$ rotations around Paulis. *arXiv e-prints*, art. arXiv:1903.12456, mar 2019.