Quantum computing in practice: fault-tolerant protocols and circuit-mapping techniques

Lao, Lingling

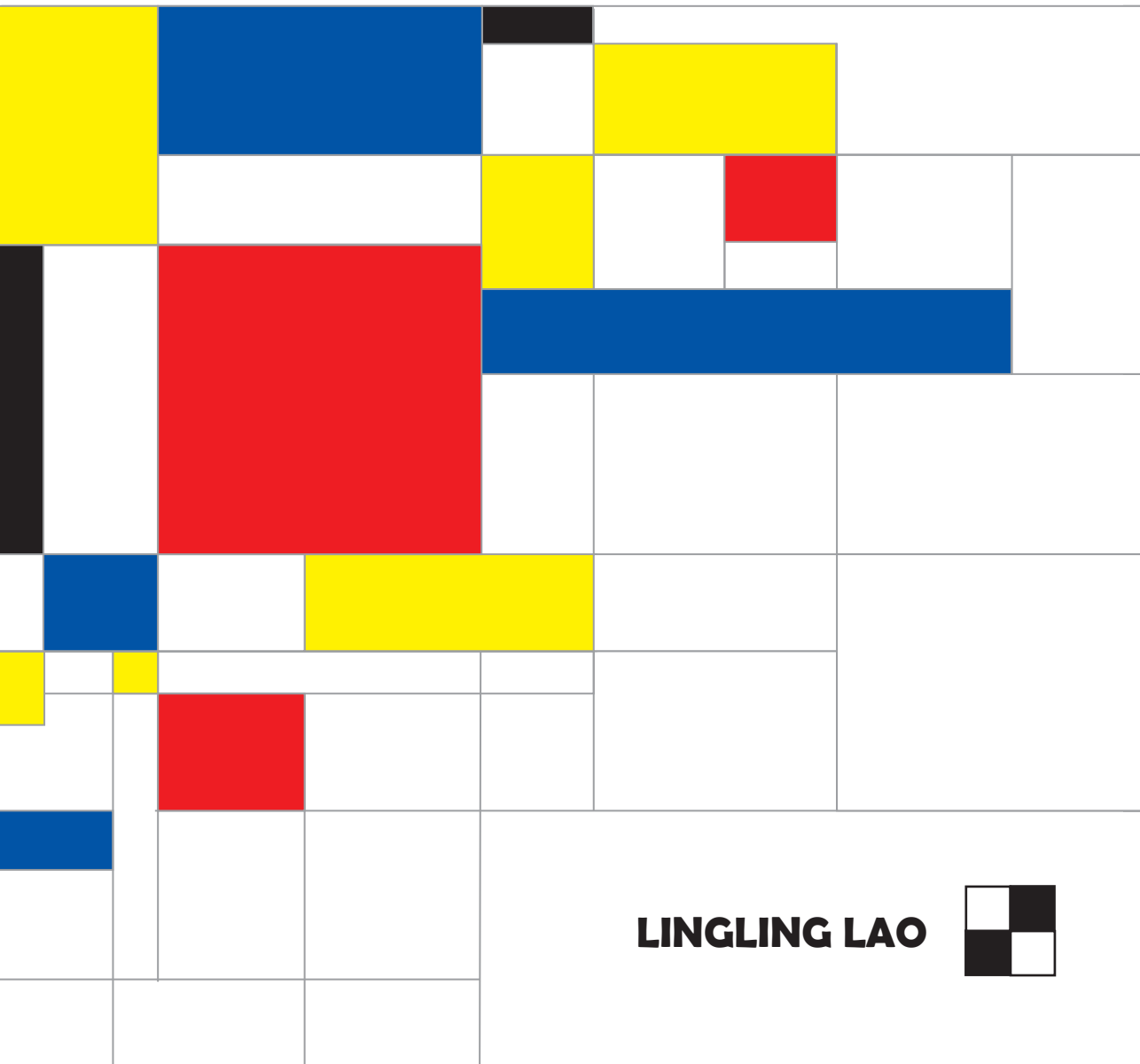
DOI
10.4233/uuid:8c60ac7e-ee2f-4090-abeb-5b5daab6ea36

Publication date
2019

Document Version
Final published version

Citation (APA)
Lao, L. (2019). *Quantum computing in practice: fault-tolerant protocols and circuit-mapping techniques*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:8c60ac7e-ee2f-4090-abeb-5b5daab6ea36

QUANTUM COMPUTING IN PRACTICE: FAULT-TOLERANT PROTOCOLS AND CIRCUIT-MAPPING TECHNIQUES



LINGLING LAO

QUANTUM COMPUTING IN PRACTICE: FAULT-TOLERANT PROTOCOLS AND CIRCUIT-MAPPING TECHNIQUES

QUANTUM COMPUTING IN PRACTICE: FAULT-TOLERANT PROTOCOLS AND CIRCUIT-MAPPING TECHNIQUES

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology
by the authority of the Rector Magnificus prof.dr.ir. T.H.J.J. van der Hagen
chair of the Board for Doctorates
to be defended publicly on
Monday 2 December 2019 at 10:00 o'clock

by

Lingling LAO

Master of Engineering in Communication and Information System,
Northwestern Polytechnical University, Xi'an, China,
born in Qiqihar, Heilongjiang Province, China.

This dissertation has been approved by the promotors.

Composition of the doctoral committee:

Rector Magnificus,	chairperson
Prof. dr. ir. K.L.M. Bertels	Delft University of Technology, promotor
Dr. C. Garcia Almudever	Delft University of Technology, copromotor
Independent members:	
Prof. dr. R.T. König	Technical University of Munich, Germany
Prof. dr. H.M. Buhrman	University of Amsterdam
Prof. dr. ir. L.M.K. Vandersypen	Delft University of Technology
Prof. dr. L. DiCarlo	Delft University of Technology
Prof. dr. B.M. Terhal	Delft University of Technology

This research was funded by the China Scholarship Council (CSC), and also supported by the Delft University of Technology.

Keywords: Fault-tolerant quantum computing, Quantum error correction, Quantum circuit mapping, Quantum computer architecture, Surface code.

Printed by: Ipskamp Printing

Cover by: Lingling Lao

ISBN:978-94-028-1838-3

An electronic version of this dissertation is available at
`http://repository.tudelft.nl/`.

To my family

CONTENTS

SUMMARY

Quantum computing promises to solve some problems that are intractable by classical computers. Several quantum processors based on different technologies and consisting of a few tens of noisy qubits have already been developed. However, qubits are fragile as they tend to decohere extremely quickly and quantum operations are faulty, making reliable computation very difficult. Moreover, quantum processors have hardware constraints such as limited qubit connectivity and shared classical control, making quantum algorithms not directly executable. This thesis focuses on some of the challenges of the implementation of quantum algorithms on near-term intermediate-scale and future large-scale quantum processors. More precisely, it investigates how to perform reliable quantum computation using fault-tolerant protocols and how to execute quantum algorithms on hardware-constrained processors using circuit-mapping techniques.

The first part of this thesis investigates the execution of large-scale quantum algorithms that requires quantum error correction (QEC) and fault-tolerant mechanisms. We focus on the rotated surface code which is one of the most promising QEC codes because of its high error threshold and simple structure that only requires 2D nearest-neighbor (NN) interactions. Firstly, the implementation of a universal set of logical operations is introduced. This includes the transversal initialization and measurement in the computational basis, the CNOT gate using lattice surgery, the Hadamard gate based on code deformation, and the S and T gates that require magic state preparation. In Chapter 3, we propose to formulate the lattice-surgery-based and code-deformation-based operations as special cases of the gauge fixing technique. We show that, by using gauge fixing, the measurement and decoding schemes for these protocols become obvious and the analysis of their fault tolerance is considerably simplified. We also numerically demonstrate the accuracy of this approach and evaluate the fault tolerance of some logical operations using the Gottesman-Knill formalism.

The construction of surface codes only requires 2D NN interactions between physical qubits, which can be directly supported by many quantum processors. However, the lattice-surgery-based implementation of CNOT gates on surface codes has specific requirements on the layouts of logical qubits. Logical qubits that need to interact and are not placed in such neighboring positions need to be moved for execution, leading to overhead in terms of both qubit resources and execution time. Reducing spatial-temporal costs becomes critical since it is beneficial to decrease the failure rate of quantum computation. To this purpose, we propose two scalable qubit plane architectures for efficiently managing qubit resources and supporting communication between logical qubits in Chapter 4. We also develop mapping passes including placement and routing of logical qubits as well as scheduling of logical operations to make lattice-surgery-based quantum circuits executable meanwhile minimizing the movement overhead and exploiting the maximum parallelism. In addition, in Chapter 5 a quantum control microarchitecture that can support quantum error correction and fault-tolerant logical op-

erations and provide flexibility and scalability is presented.

Although the surface code is promising due to its high threshold and local connectivity, it requires many physical qubits to encode one logical qubits, which is unfeasible to be applied on Noisy-Intermediate-Scale Quantum (NISQ) processors. The second part of this thesis focuses on quantum computation on NISQ processors. We first investigate quantum error correction protocols that can be used to demonstrate fault tolerance on these processors, taking into account not only the limited number of noisy qubits but also the limited qubit connectivity. In Chapter 6, we combine flag fault tolerance with qubit mapping techniques to enable an efficient flag-bridge approach which can allow FT QEC on connectivity-constrained processors with no or low qubit overhead. We further present QEC examples of the Steane code on two quantum processor topologies and numerically analyze their fault tolerance.

Additionally, these NISQ processors are also expected to be used for running some small quantum applications that can solve practical problems. To do so, quantum circuits have to be adapted to the hardware constraints imposed by quantum processors. The procedure to compile physical quantum circuits (without QEC) into ones that are executable on NISQ processors is termed as the circuit mapping problem on the physical level. Moreover, due to the noisy property of NISQ devices, it is crucial to minimize the mapping overhead in terms of number of qubits, number of gates, and circuit depth. In Chapter 7, a mapper called Qmap is developed to make quantum circuits executable with low overhead. Except the qubit connectivity and the elementary gate set, Qmap also considers the limitations imposed by the shared classical control electronics, which has not been investigated by prior works. All these device characteristics are described in a configuration file, providing the flexibility that Qmap can target different quantum processors. To evaluate the performance of the proposed mapper, we map a variety of quantum benchmarks on two different superconducting quantum processors.

SAMENVATTING

Quantum computing belooft een aantal problemen op te lossen die onoplosbaar zijn met een klassieke computer. Verschillende quantum processors gebaseerd op verschillende technologieën en bestaande uit enkele tientallen imperfecte qubits zijn al ontwikkeld. Qubits zijn echter fragiel omdat ze de neiging hebben om extreem snel te decoheren en omdat quantumoperaties imperfect zijn, waardoor een betrouwbare berekening erg moeilijk is. Bovendien hebben quantum processors hardware beperkingen zoals beperkte qubit connectiviteit en gedeelde klassieke controle, waardoor quantumalgoritmen niet direct uitvoerbaar zijn. Dit proefschrift richt zich op enkele uitdagingen van de implementatie van quantumalgoritmen op middelgrote en toekomstige grootschalige quantum processors op de korte termijn. Om preciezer te zijn, het onderzoekt hoe betrouwbare quantum computing kan worden uitgevoerd met behulp van fouttolerante (FT) protocollen en hoe quantumalgoritmen kunnen worden uitgevoerd op hardwarebeperkte processoren met circuit-mapping technieken.

Het eerste deel van dit proefschrift onderzoekt de uitvoering van grootschalige quantumalgoritmen die quantumfoutcorrectie (QEC) en FT mechanismen vereisen. We richten ons op de rotated surface code, een van de meest veelbelovende QEC-codes vanwege de hoge foutdrempel en de eenvoudige structuur die alleen 2D-naburige (NN) interacties vereist. Ten eerste wordt de implementatie van een universele set logische operaties geïntroduceerd. Dit omvat de transversale initialisatie en meting in de computational basis, de CNOT gate met behulp van lattice surgery, de Hadamard-gate op basis van code deformation en de S en T -gates die voorbereiding van een magische staat vereisen. In Hoofdstuk 3 stellen we voor om de op lattice surgery en code deformation gebaseerde operaties te formuleren als speciale gevallen van de gauge fixing techniek. We laten zien dat, door het gebruik van gauge fixing, de meet- en decoderingsschema's voor deze protocollen duidelijk worden en de analyse van hun fouttolerantie aanzienlijk wordt vereenvoudigd. We demonstreren ook numeriek de nauwkeurigheid van deze aanpak en evalueren de fouttolerantie van sommige logische operaties met behulp van het Gottesman-Knill-formalism.

De constructie van surface codes vereist alleen 2D NN-interacties tussen fysieke qubits, die direct door veel quantum processors kunnen worden ondersteund. De op lattice surgery gebaseerde implementatie van CNOT gates op surface codes heeft echter specifieke vereisten voor de lay-outs van logische qubits. Logische qubits die moeten interageren en niet in dergelijke aangrenzende posities worden geplaatst, moeten worden verplaatst voor uitvoering, wat leidt tot overhead in termen van zowel qubit-bronnen als uitvoeringstijd. Het verlagen van de ruimtelijk-temporele kosten wordt van cruciaal belang, omdat het gunstig is om het faalpercentage van quantumberekening te verlagen. Voor dit doel presenteren we twee schaalbare qubit lay-outs voor het efficiënt beheren van qubit-bronnen en voor ondersteuning van communicatie tussen logische qubits in Hoofdstuk 4. We ontwikkelen ook mapping passages inclusief plaatsing en routing van

logische qubits, evenals scheduling van logische bewerkingen om op lattice surgery gebaseerde quantumcircuits uitvoerbaar te maken en ondertussen de overhead te minimaliseren en de maximale parallelliteit te benutten. Bovendien wordt in Hoofdstuk 5 een quantumcontrolemicroarchitectuur gepresenteerd die QEC en fouttolerante logische bewerkingen kan ondersteunen en flexibiliteit en schaalbaarheid biedt.

Hoewel de surface code veelbelovend is vanwege de hoge drempel en lokale connectiviteit, vereist het veel fysieke qubits om één logische qubit te coderen, wat niet haalbaar is voor toepassing op Noisy-Intermediate-Scale Quantum (NISQ) processors. Het tweede deel van dit proefschrift richt zich op quantumberekening op NISQ-processors. We onderzoeken eerst quantumfoutcorrectieprotocollen die kunnen worden gebruikt om fouttolerantie op deze processors aan te tonen, waarbij we niet alleen rekening houden met het beperkte aantal imperfecte qubits, maar ook met de beperkte qubit-connectiviteit. In Hoofdstuk 6 combineren we flag fouttolerantie met qubit mappingtechnieken om een efficiënte flag-bridge aanpak mogelijk te maken die FT QEC mogelijk maakt op processoren met beperkte connectiviteit met geen of lage qubit overhead. We presenteren verder QEC-voorbeelden van de Steane-code op twee quantum processor typologieën en analyseren numeriek hun fouttolerantie.

Bovendien wordt verwacht dat deze NISQ-processors ook worden gebruikt voor het uitvoeren van enkele kleine quantumtoepassingen die praktische problemen kunnen oplossen. Om dit te doen, moeten quantumcircuits worden aangepast aan de hardwarebeperkingen die worden opgelegd door quantum processors. De procedure om fysieke quantumcircuits (zonder QEC) te compileren tot circuits die uitvoerbaar zijn op NISQ-processors, wordt het circuit mapping probleem op fysiek niveau genoemd. Bovendien is het vanwege de imperfecte eigenschappen van NISQ-apparaten van cruciaal belang om de mappingoverhead te minimaliseren in termen van aantal qubits, aantal gates en circuitdiepte. In Hoofdstuk 7 is een mapper genaamd Qmap ontwikkeld om quantumcircuits uitvoerbaar te maken met lage overhead. Naast qubit-connectiviteit en de elementary gate set, houdt Qmap ook rekening met de beperkingen die worden opgelegd door de gedeelde klassieke besturingselektronica, die niet is onderzocht door eerdere werken. Al deze processor eigenschappen worden beschreven in een configuratiebestand, dat de flexibiliteit biedt die Qmap op verschillende quantum processors kan richten. Om de prestaties van de voorgestelde mapper te evalueren, brengen we een aantal quantumbenchmarks in kaart op twee verschillende supergeleidende quantum processors.

1

INTRODUCTION

This chapter provides an introduction to this dissertation. We discuss the challenges one faces when building a quantum computer. Firstly, we talk about the fragility of quantum systems, leading to the challenge of making quantum computing fault-tolerant. Furthermore, we introduce the hardware limitations in real quantum processors, making quantum circuits not directly executable. Afterwards, we shortly describe the individual chapters.

1

QUANTUM computing can achieve higher performance than classical computing by exploiting superposition and entanglement. For example, it can provide exponential speedup when solving certain problems such as prime factorization [1] and polynomial speedup for many others [2] compared to their best known classical counterparts. The development of a quantum computer requires to bring together the expertise of multiple disciplines that include: physics, mathematics, electrical and computer engineering. Experimentalists are working on the fabrication of quantum hardware based on different technologies such as superconducting circuits [3, 4, 5], quantum dots [6, 7, 8], nitrogen-vacancy centers [9, 10], and trapped ions [11, 12]. Their main challenges include increasing qubit coherence time, improving gate fidelities, and achieving scalability. Since quantum systems are error-prone, another challenge is to develop quantum error correction (QEC) protocols to perform reliable quantum computation. Theorists are working on the design of QEC codes that require simple error syndrome extraction schemes and allow efficient implementation of a universal set of fault-tolerant (FT) operations. Furthermore, quantum algorithms expressed in a high-level language need to be compiled into a series of instructions that are executable on quantum processors. These processors have many hardware restrictions such as a finite number of elementary gates, limited qubit connectivity, and shared electronic control. Electrical and computer engineers focus on the definition and development of an overall quantum system architecture that bridges the gap between quantum algorithms and quantum processors.

This thesis will deal with two of the challenges when building a quantum computer: how to make quantum computing fault-tolerant and how to make quantum circuits executable on quantum processors.

1.1. MAKING QUANTUM COMPUTING FAULT-TOLERANT

Quantum algorithms can solve many problems that are intractable by classical computers. However, the Achilles' heel of quantum computing is the fragility of qubits as they tend to decohere extremely fast (loss of information due to the interaction with environment). Moreover, quantum operations are faulty, which makes reliable computation more difficult. For instance, superconducting qubits may have coherence times of tens of microseconds and gate fidelity around 99.9% [3, 4, 13, 14]. Quantum error correction schemes can make quantum computing fault-tolerant by encoding one logical qubit into many physical qubits and applying FT logical operations on these logical qubits.

The surface code (SC) is one of the most promising QEC codes. It has high tolerance to errors (error threshold is near 1% [15]) and requires a 2D structure with nearest-neighbor (NN) connectivity which is realizable in current and near-term quantum technologies [16, 17]. The performance of surface-code-based quantum memories (i.e. idling logical qubits) has been well-studied, but the performance of quantum computation (i.e. logical operations) has not been extensively analyzed by numerical simulations. Many mechanisms for performing FT operations using surface codes have been proposed, including lattice surgery [18, 19] and code deformation [20]. Characterizing these schemes in terms of their error thresholds and spatial-temporal cost is then very valuable for a practical implementation of reliable quantum computing. This thesis will discuss the FT implementation of a universal set of logical operations on rotated planar surface codes including some novel approaches like lattice rotation. Moreover, numerical

simulation results will be provided to further verify these FT schemes and analyze their fault-tolerance.

Although the surface code is suitable for implementing large-scale fault-tolerant quantum computation, it requires many physical qubits to encode a single logical qubit. This high qubit overhead makes it unfeasible for demonstrating fault tolerance in near-term quantum processors that consist of a small number of noisy qubits, known as Noisy Intermediate Scale Quantum (NISQ) processors [21]. Therefore, quantum error correction with low qubit overhead is desirable. In this thesis, we will investigate alternative QEC codes that can be potentially applied in current and near-term quantum processors. We will also design an error correction scheme to enable an efficient implementation of FT QEC in small experiments.

1.2. MAKING QUANTUM CIRCUITS EXECUTABLE

When building conventional computing platforms, the overall system architecture is defined according to different layers going from algorithms, compilers, and runtime support to the definition of an instruction set architecture, corresponding micro-architecture, and circuit implementation using a universal set of gates. A quantum computer will be defined similarly as shown in Figure 1.1 [22].

This stack of layers allows going from quantum algorithms to a specific series of signals that operate on the quantum processor as follows: first, quantum algorithms (based on the circuit model in this thesis) are described by a high-level programming language. Such a representation is normally hardware-agnostic, that is, it is not aware of the limitations imposed by the underlying quantum processors such as the elementary gate set, qubit topology, and electronic control. Therefore, a process known as mapping is required to adapt quantum circuits to the quantum processor constraints. This mapping can be performed by the compiler which outputs a series of instructions that belong to the quantum instruction set architecture. Afterwards, the microarchitecture takes these instructions as input and generates the proper signals which will be finally applied on the target qubits.

This thesis focuses on the mapping of quantum circuits. The mapping procedure is composed of several modules, including initial placement of qubits, routing of qubits, and scheduling of operations. Mapping will increase the circuit size, which in turn leads to higher failure rates. Reducing mapping overhead is crucial for implementing quantum algorithms reliably, especially in the NISQ era. In this thesis, we will develop a mapper that makes physical quantum circuits executable on NISQ devices. Furthermore, as stated above, quantum error correction is necessary for large-scale quantum computation. We will also analyze the implications of surface-code-based error correction on the circuit mapping problem. Then we will present approaches to efficiently map logical circuits onto qubit topologies with NN interactions. In addition, we will investigate microarchitectural blocks that are required to support execution of logical operations.

1.3. CHAPTER OVERVIEW

This thesis consists of eight chapters. Chapters 1 and 2 provide the motivation and background information of this thesis. Chapters 3 to 7 are the main chapters and can be

Figure 1.1: The quantum system stack.

divided into two parts. Part I (Chapters 3-5) focuses on large-scale fault-tolerant quantum computing based on surface codes. Part II (Chapters 6 and 7) discusses quantum computing in NISQ processors.

In **Chapter 2**, we first introduce the basics of quantum computing and the main ideas of quantum error correction. Then, we explain how to perform fault-tolerant error correction and computation on rotated surface codes. In addition, we also discuss the need of quantum circuit mapping passes.

In **Chapter 3**, we show the fault-tolerant implementation of a universal set of logical operations (initialization, measurement, H, S, T, and CNOT) on rotated surface codes. We formulate fault-tolerant techniques like lattice surgery and code deformation as special cases of gauge fixing. This formalism simplifies the fault-tolerance analysis of measurement-based protocols and provides clear guidance of their error correction procedure. Numerical simulation results are also provided to further verify this formalism and evaluate the fault-tolerance of these logical operations.

In **Chapter 4**, we analyze the implications of surface-code-based quantum computation on the circuit mapping problem. We propose two surface-code qubit plane architectures to support logical operations based on lattice surgery. Then, we develop a full mapping procedure for executing quantum circuits onto these architectures.

In **Chapter 5**, we introduce the architectural challenges of performing fault-tolerant quantum computing based on rotated surface codes with logical operations implemented by the techniques described in Chapter 3. We envision a control microarchitecture that can support these logical operations.

In **Chapter 6**, we investigate fault-tolerant quantum error correction schemes that can be applied on NISQ processors with small amount of qubits and limited connectivity. We propose a flag-bridge approach to enable the implementation of distance-3 QEC codes. Furthermore, we show how to perform the Steane code error correction on two different current superconducting processors and analyze their performance numerically.

In **Chapter 7**, we propose a mapper to make physical (without quantum error correction) quantum circuits executable on NISQ processors that have many hardware constraints. The constraints include the elementary gate set, qubit connectivity, and classi-

cal electronic control. We evaluate the proposed mapper on two different superconducting processors.

Finally, we conclude this thesis and discuss future research work in **Chapter 8**.

2

BACKGROUND

This preliminary chapter provides the basic notions of quantum computing. We first describe the evolution of quantum systems using the unitary matrix multiplication method. Then we introduce a more efficient representation called stabilizer formalism, which will be used in the discussion of the surface-code-based error correction and fault-tolerant computation. In addition, we also describe the procedure of mapping quantum circuits onto real quantum processors.

2

This chapter introduces the background information of this thesis. It starts with the introduction of the basics of quantum computing including quantum states and a universal set of quantum gates in Section 2.1. Since quantum systems are error prone, quantum error correction and fault tolerance are required for reliably implementing quantum algorithms. In Section 2.2, we first explain the idea of quantum error correction and introduce a family of QEC codes called stabilizer codes. Then, we focus on fault-tolerant quantum computation based on surface codes. In Section 2.3, after presenting the hardware limitations of real quantum processors, we discuss the need of mapping passes to make quantum circuits executable. For more detailed information about quantum computation and quantum error correction, we refer readers to the excellent textbook by Nielsen and Chuang [23].

2.1. BASICS OF QUANTUM COMPUTING

2.1.1. QUANTUM BITS

The basic units of information in classical computing are bits. They have exclusive states, that is, a bit can only be in either 0 or 1 state. Analogous to classical computation, quantum computation is also built upon a two-level quantum system called quantum bit or qubit. Qubits however can be in a *superposition* of basis states $|0\rangle$ and $|1\rangle$: $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ where $|\rangle$ is called a *ket* and its dual $\langle|$ is called a *bra*. α and β are called probability amplitudes and they are complex numbers. $|\alpha|^2$ and $|\beta|^2$ represent the probability of getting the result '0' or '1' respectively when measuring the qubit in the computational basis, and $|\alpha|^2 + |\beta|^2$=1. It means the state of a qubit is a unit vector in a 2-dimensional complex vector space V_2. Normally, states $|0\rangle$ and $|1\rangle$ are described by

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

A simple way of visualizing a single qubit state is using the *Bloch sphere* as shown in Figure 2.1. In this case, the probability amplitudes are given by $\alpha = \cos(\theta/2)$ and $\beta = e^{i\varphi}\sin(\theta/2)$. For example, the $|0\rangle$ and $|1\rangle$ basis states correspond to $\varphi = 0, \theta = 0$ and $\varphi = 0, \theta = \pi$, respectively.

Similarly, a state of a composite quantum system made up of n qubits can be in the superposition of all the 2^n possible states: $|\psi\rangle = \alpha_0|0\cdots00\rangle + \alpha_1|0\cdots01\rangle + \cdots + \alpha_{2^n-1}|1\ldots11\rangle$, where $\alpha_i \in \mathbb{C}$ and $\sum|\alpha_i|^2 = 1$. Note that $|0\cdots00\rangle = |0\rangle \otimes \cdots \otimes |0\rangle \otimes |0\rangle$, where $\otimes$ denotes the tensor product operator. If a composite state cannot be written as a tensor product of separate states, then it is an entangled state. In contrast, a classical n-bit system can only be in one of 2^n states at any point in time. Superposition provides a large state space, which is the essence of quantum speedup compared to classical computing. Moreover, if a quantum system can be represented by a vector state, that is, a definite state, then it is called pure state. However, sometimes a quantum state may not be known, that is, a mixture of multiple pure states $|\psi_i\rangle$ with corresponding probability p_i, then it is called mixed state. A formulation to describe both pure states and mixed states is the density matrix representation,

$$\rho \equiv \sum_i p_i |\psi_i\rangle \langle\psi_i|,$$

where ρ is positive and $tr(\rho) = 1$.

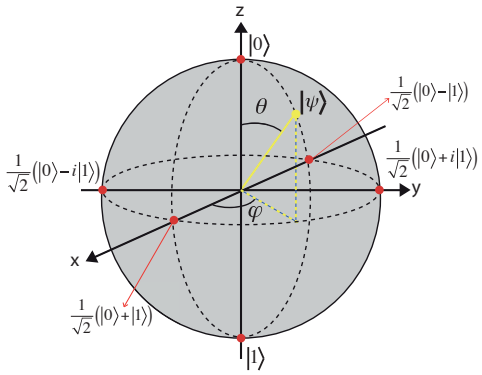


Figure 2.1: Geometrical representation of a single qubit using a Bloch sphere. The points on the surface and the interior points of the sphere correspond to pure states and mixed states, respectively.

2.1.2. QUANTUM OPERATIONS

Quantum algorithms can solve problems that are intractable by classical computers. One way to describe quantum algorithms is the quantum circuit representation, which is composed of qubits and quantum operations. In classical computing, logical operations are performed by gates such as (N)AND, (N)OR, (X)OR or NOT. These gates are described by truth tables and most of them only run forward (not reversible). In addition, any combinatorial logic function can be realized with only NAND or NOR gates, which are called universal logic gates. In quantum computing qubit states are manipulated by applying quantum operations including state initialization, quantum gates, and measurement. Any quantum gate is reversible and can be represented by a $2^n \times 2^n$ unitary matrix U, that is, $UU^\dagger = I$, where $U^\dagger$ is the *Hermitian conjugate* operator of U and n is the number of qubits it acts on. The evolution of unitary U on a quantum system represented by a vector state $|\psi\rangle$ and a density matrix ρ can be described respectively as

$$|\psi\rangle \xrightarrow{U} U|\psi\rangle$$

and

$$\rho \xrightarrow{U} U\rho U^\dagger.$$

Some of the most commonly used quantum gates are single-qubit gates and multi-qubit gates.

Single-qubit gates: Any single-qubit gate can be seen as a rotation of the Bloch sphere around some of the axis by an angle θ. For instance, Pauli X, Y, and Z gates are a rotation of the Bloch sphere around the $\hat{x}$-axis, $\hat{y}$-axis, and $\hat{z}$-axis by π (Figure 2.1), respectively. The matrix representation of single-qubit gates including Identity (I), Pauli X, Y, Z, Hadamard (H), S, T are described as follows:

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

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}, \quad T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}.$$

The group that is generated by all the Pauli matrices (I, X, Y, Z) is called the *Pauli group*. The Pauli group on one qubit is defined by

$$P_1 = \{\pm 1, \pm i\} \times \{I, X, Y, Z\}.$$

Then the n-qubit Pauli group, P_n, is generated by the tensor product of all 1-qubit Pauli group P_i, $i \in \{1, 2, \cdots, n\}$. Moreover, any two elements of the Pauli group either commute or anti-commute. If two operators A and B commute, then

$$[A, B] = AB - BA = 0.$$

If they anti-commute then

$$\{A, B\} = AB + BA = 0.$$

Where $[A, B]$ and $\{A, B\}$ are called commutator and anti-commutator, respectively. For example, the Pauli gates X, Y, Z anti-commute with each other, that is,

$$\{X, Y\} = 0, \{X, Z\} = 0, \{Z, Y\} = 0.$$

Multi-qubit gates: Commonly used multi-qubit gates include two-qubit gates such as the controlled-not (CNOT) and the controlled-phase (CZ) and three-qubit gates like the controlled-controlled-not gate (Toffoli). The CNOT gate is the equivalent of the classical XOR gate that flips the target qubit when the control qubit is $|1\rangle$, and otherwise leaves it unchanged. Similarly, the CZ gate performs a Z operation on the target qubit only if the control qubit is $|1\rangle$. The matrix representation of the controlled-not and the controlled-phase gates are:

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad \text{CZ} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}.$$

Measurements: Different from quantum gates, quantum measurements are not reversible. A measurement will project a quantum system into one of the basis states, destroying the superposition. As mentioned previously, measuring a single qubit $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ in the computational (or Z) basis will project the qubit to either $|0\rangle$ state with probability $|\alpha|^2$ or $|1\rangle$ state with probability $|\beta|^2$, yielding the corresponding measurement outcomes 0 and 1, respectively. One can also measure a single qubit in any other basis, provided that the basis states are orthonormal such that $|\alpha|^2 + |\beta|^2 = 1$. For example, the orthonormal basis states of the X basis are $|+\rangle$ and $|-\rangle$. Analogously, one can also measure multiple qubits in any orthonormal basis. A quantum measurement can be described by a collection of measurement operators, $\{M_m\}$, where m indicates the corresponding measurement outcome and $\sum_m M_m^\dagger M_m = I$. Assume this measurement is applied on a quantum state described by the density matrix ρ, then the probability of obtaining measurement outcome m is

$$p(m) = tr(M_m^\dagger M_m \rho),$$

and the state after measurement is

$$\frac{M_m^\dagger \rho M_m}{tr(M_m^\dagger M_m \rho)}.$$

2.1.3. UNIVERSALITY

In order to be able to implement any quantum circuit, a universal set of quantum gates is required. One universality construction is that any quantum gate can be exactly described by single-qubit and CNOT gates. However, not all the gates can be efficiently implemented with high fidelity in real quantum processors and only a finite number of of gates can be predefined. For instance, in the superconducting processor in [24], normally the single-qubit gates are limited to $X(Y)$-axis rotations with certain angles and the primitive two-qubit gate is the controlled-phase gate. Therefore, a discrete universal set of quantum gates is required to perform quantum computation, meaning that any arbitrary quantum gate can be approximately implemented by a finite sequence of those gates. The discrete universal gate set choice depends on the quantum hardware and QEC schemes. One of the most popular universal set of quantum gates is $\{H, S, T, \text{CNOT}\}$. Any given high-level circuit needs to be decomposed into one which only contains the gates belonging to the universal set. For instance, the Toffoli gate can be implemented by using gates in $\{H, S, T, \text{CNOT}\}$ as shown in Figure 2.2.

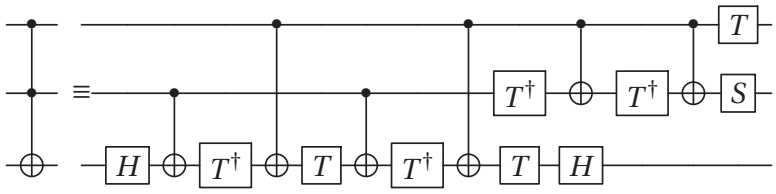


Figure 2.2: The decomposition circuit of the Toffoli gate.

2.1.4. THE STABILIZER FORMALISM

We have introduced the vector state and the density matrix methods to describe quantum states in the vector spaces. We also use the unitary operators to describe the dynamics of these states. However, a unitary that acts on a n-qubit system needs to be described by a $2^n \times 2^n$ matrix, which is difficult or impossible to simulate in classical computers for a large number n. In this section, we provide an introduction of another way of describing quantum systems, called stabilizer formalism [25]. The stabilizer formalism describes quantum states efficiently by using the operators that stabilize them (a unitary U stabilizes a pure state $|\psi\rangle$ if $U|\psi\rangle = |\psi\rangle$). For instance, the states $|0\rangle$ and $|+\rangle$ are stabilized by the operators Z and X, respectively. If a n-qubit vector space V_s is stabilized by S which is a subgroup of P_n, then every state in V_s is stabilized by every element of S and S is called the stabilizer of V_s. Note that S must be Abelian, so that any two operators in S commute with each other and $-I$ must not be in S. In addition, a stabilizer group S can be described by its generators $g_1, \cdots, g_l$, which are a subset of stabilizers that can generate any stabilizer in S by multiplication. The stabilizer group S can be written as $S = \langle g_1, \cdots, g_l \rangle$.

Furthermore, the dynamics of a quantum system can also be described efficiently by this formalism. Instead of computing how a unitary affects the vector state, we only need to compute how it transforms the stabilizer generators of this state. For example,

the Hadamard gate interchanges states $|0\rangle$ and $|+\rangle$. We can describe this transformation by

$$HXH^\dagger = Z, \quad HZH^\dagger = X.$$

Similarly, the transformation properties of other gates are as follows:

$$SXS^\dagger = Y, \quad SZS^\dagger = Z;$$

$$\text{CNOT}X_1\text{CNOT}^\dagger = X_1X_2,\ \text{CNOT}X_2\text{CNOT}^\dagger = X_2,\ \text{CNOT}Z_1\text{CNOT}^\dagger = Z_1,\ \text{CNOT}Z_2\text{CNOT}^\dagger = Z_1Z_2.$$

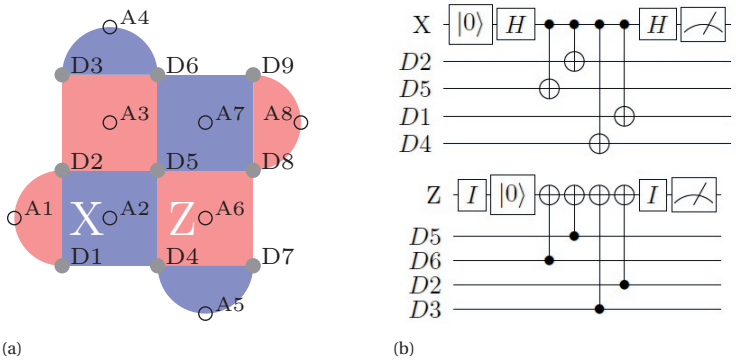
The gate set $\{H, S, \text{CNOT}\}$ generates a popular gate group called *Clifford group*. The Clifford group is the normalizer of the Pauli group, that is, it transforms each element of the Pauli group to an element of the Pauli group under conjugation. A quantum computation which only involves Clifford gates, stabilizer state preparation, and computational basis measurement can be efficiently simulated by classical computers using the stabilizer formalism according to the *Gottesman-Knill theorem* [25]. We will use the stabilizer formalism to describe some quantum error correction codes and quantum computation in this thesis.

2.2. QUANTUM ERROR CORRECTION AND FAULT-TOLERANT COMPUTATION

2.2.1. THE STABILIZER CODES

The idea of quantum error correction is to encode a logical qubit into several physical qubits called *data qubits* such that the logical qubit is reliable even though these physical qubits are imperfect. The possible errors are detected by performing error syndrome extraction or measurement (ESM) using additional *ancilla qubits*. The measurement outcomes of ancillas are called syndromes. A decoding algorithm is applied to identify highly probable errors based on the observed syndromes. The number of errors that can be detected and corrected is determined by the code distance d which is defined by the minimum number of physical operations required to perform a logical operation. Many kinds of QEC codes have been developed, such as concatenated codes like the Steane code [26], subsystem codes like Bacon-Shor codes [27], and topological codes like surface codes [15] derived from Kitaev's toric code [28].

The Steane code, Bacon-Shor codes, the surface code and many other QEC codes belong to a code class called stabilizer codes [29]. A stabilizer code that encodes k logical qubits into n physical qubits is defined using a stabilizer group S that has $n-k$ independent generators, $S = \langle g_1, \cdots, g_{n-k} \rangle$. It means the states of all data qubits are stabilized by S and form a codespace $T = \{|\psi\rangle \mid M|\psi\rangle = |\psi\rangle, |\psi\rangle \in (\mathbb{C}^2)^{\otimes n}, \forall M \in S\}$. We denote such a code $C(S)$. A logical X operator and a logical Z operator of $C(S)$ anti-commute with each other, but they commute with all the stabilizers. Moreover, error syndrome extraction is performed by measuring all stabilizer generators. If an error $E \in P_n$ anti-commutes with a stabilizer, then it can be detected since the measurement result of this stabilizer will be nontrivial. If an error E belongs to S, then it is harmless since it does not change the code space. However, if an error E commutes with all the stabilizers but is not in S, then it is a bad error and cannot be detected. To summarize, for a stabilizer code $C(S)$ and a set of Pauli errors $\{E_j\}$, $\{E_j\}$ is a correctable error set if $E_j^\dagger E_k \notin N(S) - S$, where



(a) (b)

Figure 2.3: (a) The qubit layout for the rotated distance-3 surface code where data qubits are on the vertices (solid circles) and ancilla qubits are on the plaquettes (open circles). The purple (pink) squares and semi-circles represent stabilizers of the form $X(Z)^{\otimes 4}$ and $X(Z)^{\otimes 2}$, respectively. (b) Circuits for measuring X- and Z-stabilizers ($X_{D5,D2,D4,D1}$ and $Z_{D6,D3,D5,D2}$).

$N(S) = \{g \mid gs = sg, s \in S\}$ is the normalizer of S. In addition, the distance of the code $C(S)$ is the minimum weight of an operator in $N(S) - S$.

2.2.2. THE SURFACE CODE

The surface code is a topological stabilizer code implemented on a 2-dimensional array of physical qubits with only NN interactions as shown in Figure 2.3a. It consists of two types of qubits, data qubits (solid circles) for storing computational information, and ancilla qubits (open circles) used to perform stabilizer measurement. Each ancilla qubit is coupled to 2 or 4 data qubits, depending on the stabilizer size. Each data qubit interacts with 2 or 4 ancillas in two differently colored plaquettes, which correspond to two types of stabilizers, X-stabilizers for detecting Z errors, and Z-stabilizers for detecting X errors. For instance, an error X_{D2} will result in -1 syndromes on stabilizers $Z_{D1,D2}$ and $Z_{D2,D3,D5,D6}$. The circuits to perform X- and Z-stabilizer measurement are shown on Figure 2.3b. We define a SC cycle as the interval between the starting points of two consecutive ESM. In principle, one only needs one SC cycle to detect data errors. However, errors can also occur on measurement qubits and may change the syndromes of measured stabilizers. At least d SC cycles are required to detect both data (in space) and measurement (in time) errors [30]. Decoding algorithms such as minimum weight perfect matching [31, 32, 33], re-normalization group [34], tensor networks [35], and neural networks [36, 37] can be used to identify the possible errors which lead to observed syndromes with high probability [30]. Rather than physically performing these corrections which will introduce more errors to the quantum system, errors can be tracked by classical control software [30] such as 'Pauli Frame' [38].

In surface codes, there are two main ways of encoding a single logical qubit, using a *planar* or a *defect* approach. For the planar approach, an entire patch is used to encode just one logical qubit as shown in Figure 2.3a. The defect logical qubit is realized by creating two Z-cut holes (turn off Z-stabilizers) or X-cut holes (turn off X-stabilizers), more details can be found in [15]. Planar surface codes require fewer physical qubits to

encode one logical qubit for the same code distance compared to the defect approach. This is why current quantum technologies in which qubits are scarce resources are pursuing the realization of planar SC quantum hardware [39]. This thesis will only focus on planar surface codes.

2.2.3. FAULT-TOLERANT QUANTUM COMPUTATION

Quantum error correction could recover quantum states from errors, which is useful for storing and transmitting quantum information. However, it is not sufficient for realizing reliable quantum computers since the encoding and computing processes will inevitably introduce errors that can propagate and accumulate through quantum operations, causing the whole computation to fail. Therefore, fault-tolerance is necessary for the implementation of both quantum error correction and quantum computation. A procedure is fault-tolerant if one single error cannot spread more than one error to each code block. The most robust and efficient method to fault-tolerantly perform logical operations is the transversal implementation, that is, performing bit-wise or pair-wise physical operations on the data qubits of the code. In planar SC, initialization, measurement, Pauli gates, and H can be implemented transversally. For example, the $\overline{X}$ and $\overline{Z}$ on SC-17 (Figure 2.3a) can be realized by performing $X_{D1,D2,D3}$ and $Z_{D1,D4,D7}$. In principle, a FT CNOT gate between two planar surface code qubits can be implemented transversally. However, this transversal CNOT requires a 3D architecture, which is not realizable in current quantum technologies that have 2D architectures with nearest-neighbor qubit connectivity. Alternatively, a FT CNOT gate can be achieved by a technique called lattice surgery [18] that complies to the 2D NN interaction constraint. The detailed implementation of lattice-surgery-based CNOT gates will be introduced in Chapter 2.

To complete a universal set of gates on surface codes, one also needs to implement a logical T gate in surface codes. The FT implementation of a T gate in surface code requires ancillary qubits prepared in specific states called magic states. Similarly, a logical S gate also has to be implemented indirectly. The circuits to perform a logical S and a logical T using magic states are shown in Figure 2.4 [40]. Magic states $|Y\rangle$ and $|A\rangle$ need to be prepared for S and T, respectively. One can prepare these magic states by using an injection procedure that can be implemented by measurement-based operations [18, 19]. However, the state injection procedure is not fault-tolerant and injected states need to be purified by another procedure called state distillation [40, 41, 42, 43, 44]. However, magic state distillation is a non-deterministic procedure, it must be repeated until the measurement results indicate one state is successfully purified. The success probability of distillation (P_s) depends on the logical error rate on input states (p), and once it succeeds, the infidelity of the accepted state will be suppressed (e.g., the infidelity is suppressed to $O(p^3)$ in [15]). Moreover, multiple rounds of successful distillation may be required to achieve the desired state infidelity $O(p^n)$. Therefore, magic state distillation is extremely resource- and time-consuming. In fact, it may be the most expensive procedure in quantum computing.

(a) (b)

Figure 2.4: The circuits to perform a logical S (a) and a logical T (b).

2.3. MAPPING OF QUANTUM CIRCUITS

2.3.1. HARDWARE CONSTRAINTS

When adopting the circuit model as a computational model, quantum algorithms can be represented by quantum circuits consisting of qubits and gates. These circuits are normally hardware-agnostic and cannot be directly executed on real quantum processors that have many hardware constraints, including the followings:

- Elementary gate set: To enable the implementation of any quantum circuit, a universal set of quantum operations including qubit preparation and measurement, single-qubit rotations, and multi-qubit gates, are required. However, generally only a finite number of quantum gates with relatively high fidelity are predefined on real quantum processors, composing the elementary gate set. For instance, X and Y rotations with ± 45, ± 90 and ± 180 degrees, and control-phase gates are normally predefined on the superconducting processor in [24]. Therefore, any given quantum circuits need to be decomposed into one which only consists of the predefined elementary gates.

- Qubit connectivity: One of the most promising physical qubit layouts that is being pursued for many quantum technologies like superconducting qubits [16, 45] and quantum dots [17, 46], is a 2D architecture that only allows nearest-neighbor interactions. Non-neighboring qubits need to be moved to be adjacent for interacting, which can be implemented by either physical movement of qubits (e.g., ion shuttling in ion trap quantum processors [47]) or virtual movement through quantum operations (e.g., exchanging the states of two qubits by SWAP gates). These movement operations will result in an overhead in the number of qubits and gates as well as in the execution time (latency) of the circuit, decreasing the reliability of a given quantum algorithm.

- Classical control: classical electronics are required for controlling and operating the qubits. Using a dedicated instrument per qubit is not scalable and very expensive approach. For a practical fabric of quantum processors, it is necessary to simplify classical control hardware, which restricts the control signals for implementing quantum operations and in turn limits the possible parallelism of quantum operations. For example, three microwave frequencies are used for single-qubit control and eight detuning frequencies are used for two-qubit gates in a superconducting surface-code fabric [39].

2.3.2. THE MAPPING PROCEDURE

All these constraints may vary between different quantum technologies and different processors. To comply with these hardware constraints when implementing quantum algorithms, a procedure called mapping of quantum circuits is therefore required. The mapping includes placement and routing of qubits as well as scheduling of operations. That is, it places virtual qubits in the circuit to hardware qubits in the processor; it finds routing paths to move non-neighboring qubits to be adjacent for interaction; and it schedules quantum operations to exploit available parallelism with respect to the data dependencies. The implementation details of each mapping module will be explained in Chapter 4 and Chapter 7. The mapping of quantum circuits will result in an overhead in terms of the number of operations, circuit depth, and circuit latency (the real execution time of a circuit). Therefore, reducing the overhead caused by the mapping procedure will be crucial to improve the reliability of quantum algorithms.

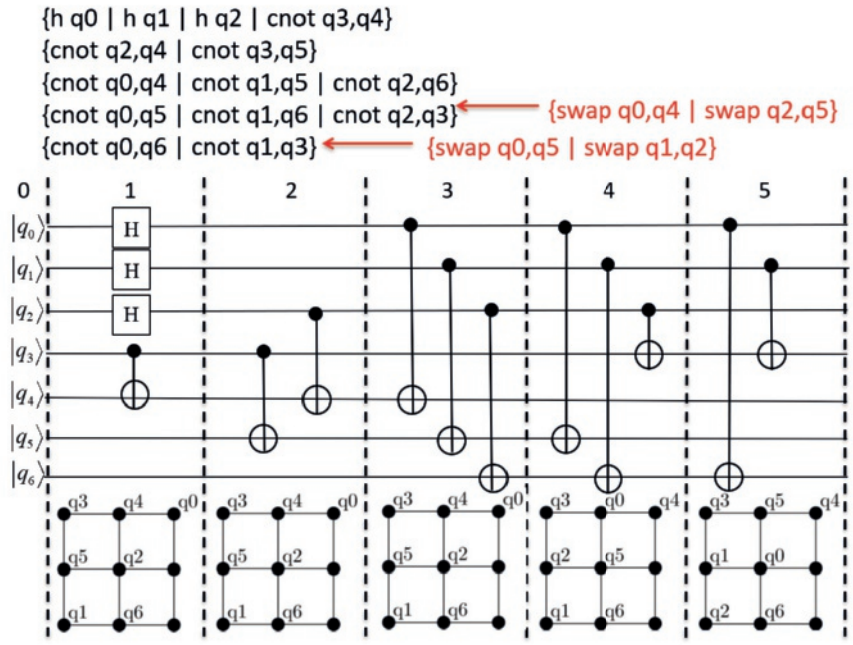


Figure 2.5: Mapping of a quantum circuit onto a 2D NN architecture, the circuit and its QASM description are shown on the top. Operations in the same bracket (QASM code) correspond to the gates that can be performed in parallel (separated by dashed lines in the circuit).

An example of mapping a (physical) quantum circuit onto a 2D NN architecture consisting of 3×3 qubits, including placement and routing of qubits as well as scheduling operations, is shown in Figure 2.5. In this example only the qubit connectivity constraint is considered and it assumes that all the operations take one time-step. The given circuit is first scheduled in an as-soon-as-possible manner as shown in the middle of Figure 2.5 where the operations between two dashed lines are performed at the same timestep. It

will take five timesteps to execute this circuit if the connectivity constraint is not considered. This circuit is also described by a quantum assembly (QASM) language [23, 48]. The QASM description of the pre-scheduled circuit is shown on the top, where the operations inside the same bracket are performed in parallel. Furthermore, an initial placement of these seven qubits is shown in the layout at timestep 1. All the operations scheduled at timesteps 0 to 3 can be directly performed on this layout. However, in order to perform the CNOT gates on (q0, q5) and (q2, q3) at timestep 4, two SWAP gates (in red) need to be inserted to route qubits to be adjacent. The layouts of qubit change during the routing procedure as shown on the bottom of Figure 2.5. In total, four SWAP gates are inserted for mapping this circuit, increasing the number of operations and the circuit timesteps from 14 to 18 and from 5 to 7, respectively.

As we will show in this thesis, the mapping procedure is required not only for running quantum algorithms in current and near-term processors where no QEC is used (Chapter 7), but also for performing fault-tolerant quantum computation using QEC in large-scale devices. This is because 1) the syndrome extraction circuits of a quantum error correction code have requirements on the connectivity between physical qubits (e.g. the surface code requires 2D NN interactions); 2) the fault-tolerant implementation of logical operations causes extra constraints at the logical level (e.g. a lattice-surgery-based CNOT on rotated planar surface codes requires the logical qubits to be placed in specific neighboring positions.). The mapping of surface-code-based quantum circuits will be discussed in Chapter 4.

3

FAULT-TOLERANT COMPUTATION BASED ON SURFACE CODES

The large-scale execution of quantum algorithms requires basic quantum operations to be implemented fault-tolerantly. The most popular technique for accomplishing this, using the devices that can be realized in the near term, uses stabilizer codes which can be embedded in a planar layout. The set of fault-tolerant operations which can be executed in these systems using unitary gates is typically very limited. This has driven the development of measurement-based schemes for performing logical operations in these codes, known as lattice surgery and code deformation. In parallel, gauge fixing has emerged as a measurement-based method for performing universal gate sets in subsystem stabilizer codes. In this chapter, we show that lattice surgery and code deformation can be expressed as special cases of gauge fixing, permitting a simple and rigorous test for fault-tolerance together with simple guiding principles for the implementation of these operations. We demonstrate the accuracy of this method numerically with examples based on the surface code, some of which are novel.

The results of this chapter have been published in C. Vuillot*, L. Lao*, B. Criger, C. G. Almudever, K. Bertels, B. Terhal, *Code deformation and lattice surgery are gauge fixing*, New Journal of Physics **21**, 033028 (2019).
*Those authors contribute equally to this paper. In particular, L. Lao was involved in all the conceptual understanding of the gauge fixing idea and executed the numerical work.

3.1. INTRODUCTION

QUANTUM computers can implement algorithms which are much faster than their classical counterparts, with exponential speedup for problems such as prime factorization [1], and polynomial speedup for many others [2]. The main obstacle to constructing a large-scale quantum computer is decoherence, which partially randomizes quantum states and operations. Although state-of-the-art coherence times are now appreciably longer than gate times [4, 3], they remain too short for useful quantum computation.

To counter the effect of decoherence on quantum states which are stored or manipulated imperfectly, we can encode logical qubit states into several physical qubits, and perform non-destructive multi-qubit measurements of the resulting system to extract information about which errors have occurred, called the *syndrome*. The spaces of multi-qubit states used to encode these logical states are called *quantum error-correcting codes*, and their ability to correct errors is measured by the *distance d*, which is the number of independent errors (or error *weight*) necessary to alter the state of the logical qubits without being detected. In order to use one of these codes in practice, it is also necessary to account for the effect of decoherence on operations. For example, a syndrome measurement may involve a sequence of entangling gates, and the error caused by a faulty gate on a small set of qubits in the beginning of the circuit may propagate onto many qubits, producing a high-weight error, increasing the likelihood of a logical error. Measurement results can also be corrupted by decoherence, so syndrome extraction often has to be repeated. In order to prevent error propagation during repeated measurement, syndrome extraction circuits must be designed such that a small number of faults (from imperfect gates or memory errors on data qubits) will result in a small number of errors on the physical qubits, which can be corrected using noisy syndromes. Given a family of codes of different distances, we can determine a *threshold error rate*, the rate beneath which codes with higher distance produce lower logical error probabilities.

Several such families of quantum error-correcting codes have been developed, including concatenated codes [26, 49], subsystem codes such as Bacon-Shor codes [27], and 2D topological codes. The most prominent 2D topological codes are surface codes [15] derived from Kitaev's toric code [28], which we will focus on in the remainder of this manuscript. 2D topological codes can be implemented using entangling gates which are local in two dimensions, allowing fault-tolerance in near-term devices which have limited connectivity. In addition, 2D topological codes generally have high fault-tolerant memory thresholds, with the surface code having the highest at $\sim 1\%$ [50].

These advantages come at a cost, however. While other 2D topological codes permit certain single-qubit logical operations to be implemented *transversally*, the surface code does not. In addition, the constraint that computation be carried out in a single plane does not permit two-qubit physical gates to be carried out between physical qubits in different code blocks, precluding the two-qubit gates which, in principle, can be carried out transversally.

These two restrictions have led to the design of measurement-based protocols for performing single- and two-qubit logical gates by making gradual changes to the underlying stabilizer code. Measurement-based protocols that implement single-qubit gates are typically called *code deformation* [20], and protocols that involve multiple logical

qubits are usually called *lattice surgery* [18]. A separate measurement-based technique, called *gauge fixing* [51], can be applied to *subsystem codes*, which have operators which can be added to or removed from the stabilizer group as desired, the so-called *gauge operators*. During gauge fixing, the stabilizer generators of the subsystem code remain unchanged, and can be used to detect and correct errors; so decoding is unaffected by gauge fixing. This is in contrast to code deformation and lattice surgery, where it is not *a priori* clear which measurement results to incorporate into decoding, or how to process them. Recently, many different code deformation and lattice surgery techniques have been devised, most of which use tailor-made analysis or decoding techniques, see e.g. [19, 52, 53, 54, 55, 56, 57, 58].

In this chapter, we phrase existing lattice surgery and code deformation protocols as special cases of gauge fixing, showing that the underlying subsystem code dictates the fault-tolerance properties of the protocol. This perspective can simplify the analysis of new measurement-based protocols, provided that they are based on stabilizer codes whose distances can be easily calculated. Also, knowing the stabilizer of the underlying subsystem code results in clear guidelines for decoding using the measurement results produced by such a protocol.

The remainder of this chapter is organized as follows. In Section 3.2, we review the ideas behind code deformation and lattice surgery. In Section 3.3, we review the formalism of gauge fixing. Following this, in Section 3.4, we formulate lattice surgery and code deformation operations as gauge fixing, demonstrating that fault-tolerant code deformation protocols are in fact based on high-distance subsystem codes. We also show this explicitly using both well-known and novel protocols. In Section 3.5, we numerically determine the performance of these protocols. We conclude and discuss potential future research in Section 3.6.

In all figures in this chapter, qubits are located on the vertices of the drawn lattice. We refer to the local generators of the stabilizer group of the surface code as *stabilizers* or *checks*. In the figures, black regions signify X-stabilizers and light gray regions Z-stabilizers, with no stabilizers measured on white plaquettes.

3.2. CODE DEFORMATION AND LATTICE SURGERY

3.2.1. CODE DEFORMATION

Code deformation is a technique to convert one code into another by making a series of changes to the set of stabilizer generators to be measured in each round of error correction.

Typically, these protocols use ancilla prepared in entangled and/or encoded states as a resource. Also, a typical code deformation sequence proceeds gradually, first expanding the code into a large intermediate code by entangling the original code block with the ancilla, then disentangling some of the qubits (which may include some or all of the original data qubits), producing a final code which can then be used for further computation. The initial and final code may differ in their logical operators, in which case the deformation performs a logical operation. Also, the initial and final code may differ in their position or orientation within a larger quantum computer.

For example, consider the proposed fault-tolerant procedure for lattice rotation of

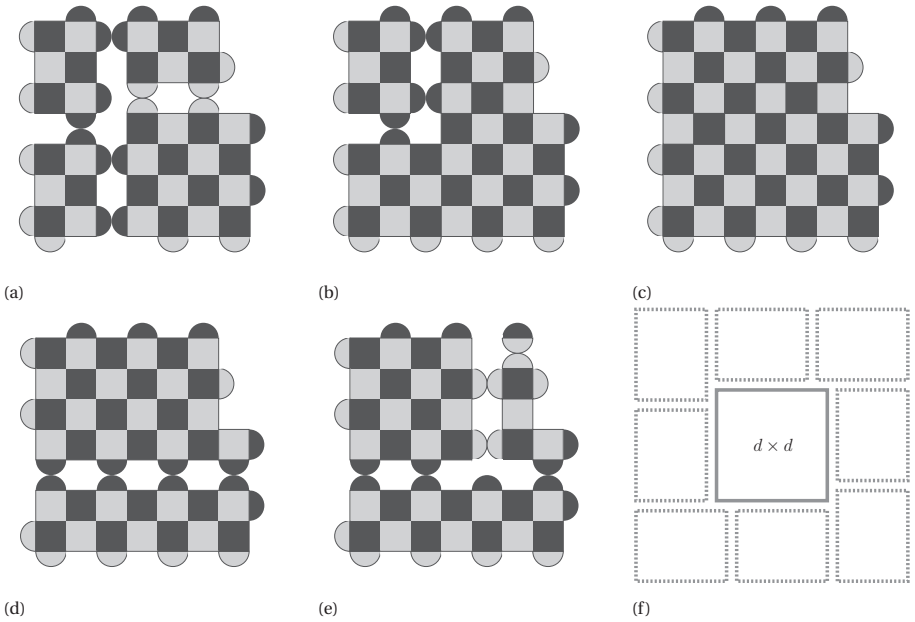
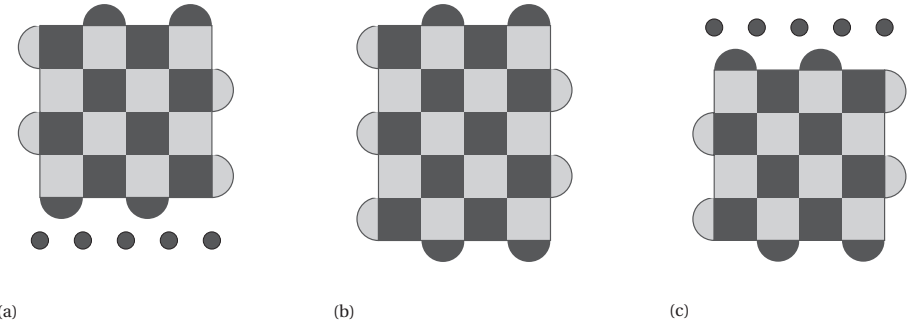


Figure 3.1: Fault-tolerant procedure for rotating a surface code by 90° and reflecting it about the x axis (see [18, Figure 10] for the corresponding protocol using smooth/rough boundaries). (a) Initial layout where the 5×5 lattice is to be rotated, the three 3×4 patches are ancillas in fixed states, fully specified by the stabilizers shown. (b) Intermediate lattice, this step is required to expand the lattice fault-tolerantly. (c) Fully expanded lattice. (d) and (e) Splitting operations performed to shrink the lattice. (f) By using the two steps from (a) to (c) at the same time on all corners, one can grow a lattice from distance d to $3d-4$. The surrounding ancillary patches have $(d-2) \times (d-1)$ qubits each.

surface codes shown in Figure 3.1, similar to the one presented in [30]. One can see five steps which gradually modify the surface code patch starting at the bottom right of Figure 3.1a and ending at the top left of Figure 3.1e in a different orientation. First, three ancillary patches are prepared in fixed states, and placed near the upper left corner of the target patch. Then, the patch undergoes a two-step growing operation, followed by a two-step shrinking operation. Advancing one step is done by measuring the operators corresponding to the new stabilizers, some of which anti-commute with the old ones. Measurement of these new stabilizers will return ± 1 values at random. This means that additional corrections, unrelated to errors that may have occurred, are needed in order to enter the new code space (the mutual $+1$-eigenspace of all new stabilizers). Moreover, to account for noisy operations, one must simultaneously perform error correction. After one is confident that the encoded state is in the new code space, one can proceed to the next step.

In Section 3.4, we will demonstrate that, following these five steps, one can fault-tolerantly protect the logical information at all times with a distance-5 code. We also show that the distance would be reduced to 3 if one were to omit step (Figure 3.2b), going directly from (Figure 3.2a) to (Figure 3.2c), as one would do when directly adapting the



(a) (b) (c)

Figure 3.2: A procedure to flip a lattice using code deformation. (a) The lattice to be flipped, and the physical qubits prepared in $|+\rangle$ states. (b) The flip operation is realized by merging the original lattice with the physical qubits below. (c) Subsequently measuring the physical qubits at the top in the X basis finishes the flip operation.

established surface code rotation method from [18] to rotated surface codes.

This lattice rotation followed by the lattice flip in Figure 3.2 are useful for performing a transversal Hadamard gate. The transversal Hadamard gate on a surface code patch, performed by applying a Hadamard gate on each qubit, interchanges X and Z plaquettes. This code transformation can be undone by a lattice rotation, followed by a lattice flip. Moreover, part of this rotation procedure can be used to grow a code with distance d to a code with distance $(3d-4)$ in two steps by simultaneously growing all corners, see Figure 3.1f.

This type of code deformation does not, in itself, perform logical operations, but can be used to move patches of code or to convert between codes where different gates are transversal [53]. Other code deformation procedures such as moving holes or twists do perform unitary logical Clifford operations [55, 59, 60]. In the next section, we present another similar procedure which executes a logical measurement.

3.2.2. LATTICE SURGERY

Lattice surgery is a particular measurement-based procedure that acts non-trivially on logical information. By going through two steps of deformation, it implements a joint measurement of logical operators, typically $\overline{X}_1\overline{X}_2$ or $\overline{Z}_1\overline{Z}_2$, where $\overline{X}_j$ and $\overline{Z}_j$ denote the logical operators of the logical qubit j. We will focus on the $\overline{Z}_1\overline{Z}_2$ measurement and review the protocol used for the surface code [18, 19].

Consider two patches of $L \times L$ rotated surface code, as in Figure 3.3a. Each has a $\overline{Z}$ along the boundary which faces the other patch. In the *merge* step, one measures the intermediary Z-plaquettes (in pink in Figure 3.3b). These plaquettes are such that the product of all outcomes is the outcome of the $\overline{Z}_1\overline{Z}_2$ measurement, but any subset of these outcomes produces a random result when multiplied together. This ensures that the only non-stabilizer operator whose eigenvalue can be inferred from these measurements is $\overline{Z}_1\overline{Z}_2$. These measurements do not commute with the weight-2 X stabilizers at the joint boundary (in Figure 3.3a). The Gottesman-Knill theorem [25] prescribes how to update the stabilizer after such measurements, namely we only retain elements in

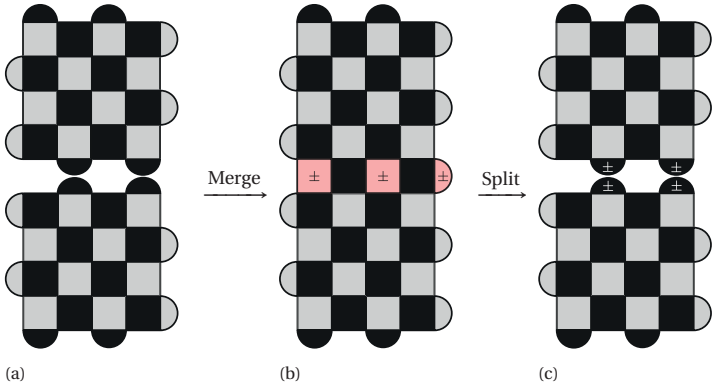


Figure 3.3: Lattice surgery for the rotated surface code. Grey plaquettes show Z-stabilizers, black plaquettes represent X-stabilizers. A '±' label indicates a random sign for the corresponding plaquette in the stabilizer group. (a) Initial layout, two rotated surface codes. (b) The merged lattice, which is a surface code with random ± signs on the newly-measured (red) plaquettes. (c) The *split* lattices, in which the original stabilizers are measured again. Random ± signs are produced on the boundary X-stabilizers.

the original stabilizer group which do commute with the newly measured stabilizers. This implies that the code becomes a $2L \times L$ patch of surface code, apart from some minus signs on the newly-measured Z-checks. This merge step is very similar to the rotation presented before, except that some logical information is gained in the process and the additional corrections which fix the state into the new code space may involve one of the original logical operators (when the number of intermediary plaquettes with -1 eigenvalues is odd). To finish the protocol, the original code space must be restored by performing a *splitting* operation, measuring the original stabilizers of the two separate patches instead of the intermediary Z-plaquettes. Those Z-plaquettes, as in the merge step, anticommute with the boundary X-stabilizers, and will be removed from the stabilizer group. Their product, equal to $\overline{Z}_1\overline{Z}_2$, does commute, and will remain as a stabilizer of the final state. In addition, the boundary X-plaquettes will have random ± signs which are perfectly correlated between facing pairs. Therefore, one can eliminate these ± signs by applying some of the former stabilizers (those supported on the intermediary Z-plaquettes).

One can check that depending on the outcome (± 1) of the logical $\overline{Z}_1\overline{Z}_2$ measurement, the merge and split operations, respectively $M_\pm$ and $S_\pm$ can be expressed as

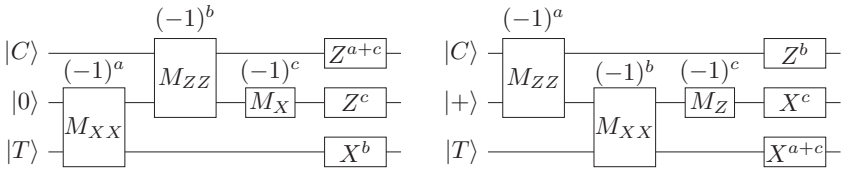
$$M_+ = |\overline{0}\rangle\langle\overline{00}| + |\overline{1}\rangle\langle\overline{11}|, \qquad S_+ = |\overline{00}\rangle\langle\overline{0}| + |\overline{11}\rangle\langle\overline{1}|, \tag{3.1}$$

$$M_- = |\overline{0}\rangle\langle\overline{01}| + |\overline{1}\rangle\langle\overline{10}|, \qquad S_- = |\overline{01}\rangle\langle\overline{0}| + |\overline{10}\rangle\langle\overline{1}|. \tag{3.2}$$

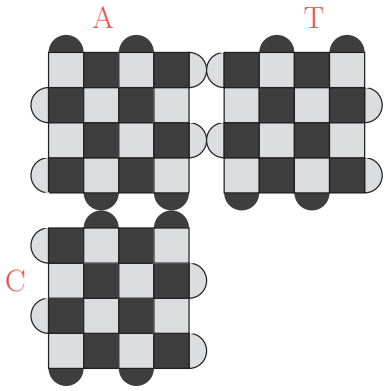
They are related to the projections, $P_\pm$, onto the ± 1 eigenspace of $\overline{Z}_1\overline{Z}_2$ by composition:

$$P_+ = S_+ \circ M_+, \qquad P_- = S_- \circ M_-.$$

In particular, lattice surgery allows us to implement the measurement-based CNOT gate [61] in a 2D layout with only local operations as shown in Figure 3.4. We note that a



(a)



(b)

Figure 3.4: (a) Two equivalent measurement-based circuits for the CNOT gate. (b) The qubit layout for a CNOT gate between two surface-code qubits. C is the control qubit, T is the target qubit, and A is a logical ancilla.

more general set of operations which can be implemented by lattice surgery can be constructed using the relation between the merge and split operations considered here and the three-legged nodes of the ZX-calculus [62]. For the purposes of this work, however, we will limit our discussion to CNOT gates.

3.3. GAUGE FIXING

Gauge fixing [51] is an approach which has been used to implement universal fault-tolerant gate sets in *subsystem codes* [63]. A subsystem code is equivalent to a stabilizer code in which some of the logical qubits are not used to carry any logical information. These logical qubits are called *gauge* qubits and they can be acted on or measured without disturbing the states of the other logical qubits, which are used to store and process quantum information. Then, one way to formally define a subsystem code, C, is to define a subgroup of the Pauli group, called the *gauge group* $\mathcal{G}$, containing all the Pauli stabilizers as well as the Pauli operators defining the gauge qubits. This subgroup is non-Abelian as it contains anti-commuting Pauli operator pairs which represent the gauge qubit logical operators. The stabilizer group, $\mathcal{S}$, can be derived from $\mathcal{G}$ as its center, denoted Z(·), i.e. containing all elements in $\mathcal{G}$ which mutually commute

$$\mathcal{S} = \mathrm{Z}(\mathcal{G}) = \mathcal{C}(\mathcal{G}) \cap \mathcal{G}, \tag{3.3}$$

where $\mathscr{C}(\mathscr{G})$ denotes the centralizer of $\mathscr{G}$ in the Pauli group, i.e. all elements in the Pauli group with commute with all elements in $\mathscr{G}$. Elements in $\mathscr{G}$ which are not in $\mathscr{S}$ are the Pauli operators acting non-trivially on the gauge qubits: this is the set of gauge operators $\mathscr{L}_g$

$$\mathscr{L}_g = \mathscr{G} \setminus \mathscr{S}. \tag{3.4}$$

Following this, one can define operators for the actual logical qubits which by definition are elements in $\mathscr{C}(\mathscr{S}) \setminus \mathscr{S}$. If these operators act trivially on the gauge qubits, we call these *bare* logical operators. Bare logical operators can be multiplied by elements in $\mathscr{L}_g$ to become *dressed* logical operators which also act on the gauge qubits. We can write

$$\mathscr{L}_{\text{bare}} = \mathscr{C}(\mathscr{G}) \setminus \mathscr{G}, \qquad \mathscr{L}_{\text{dressed}} = \mathscr{C}(\mathscr{S}) \setminus \mathscr{G}. \tag{3.5}$$

Note that with this definition we have, $\mathscr{L}_{\text{bare}} \subset \mathscr{L}_{\text{dressed}}$. The distance of the subsystem code C is the smallest weight of any of its logical operators,

$$d_C = \min_{\ell \in \mathscr{L}_{\text{dressed}}} \text{wt}(\ell). \tag{3.6}$$

One advantage of subsystem codes is that to measure stabilizers, one is free to measure any set of checks in the gauge group as long as this set generates the stabilizer group. By measuring elements in the full gauge group, one can put the gauge qubits in specific states, permitting different sets of transversal logical gates. This act of putting the gauge qubits in a specific state is called *gauge fixing*. The idea is to measure a commuting subset of gauge operators (all the Z-type gauge operators, for example), obtaining ± 1 outcomes and applying the anticommuting, or *conjugate* partner operator (an X-type gauge operator in the example), wherever a -1 outcome has been obtained. In the example, this would fix all gauge qubits to the $|0\rangle$ state. While the gauge is fixed in this way, the Z-type gauge operators become elements of the stabilizer group, so $\mathscr{S}$ is augmented to some larger Abelian subgroup of $\mathscr{G}$. In Section 3.7.1, we will show an example of how code conversion between the $[\![7, 1, 3]\!]$ Steane code to the $[\![15, 7, 3]\!]$ Reed-Muller code can be viewed as gauge fixing.

3.4. FAULT-TOLERANCE ANALYSIS WITH GAUGE FIXING

In this section, we will briefly show that one can use gauge fixing to analyze the fault tolerance of both code deformation and lattice surgery operations. A more detailed analysis can be found in [64].

3.4.1. FAULT-TOLERANCE OF CODE DEFORMATION

We consider the QEC codes before and after an operation, represented by C_{old} and C_{new}, with stabilizer groups $\mathscr{S}_{\text{old}}$ and $\mathscr{S}_{\text{new}}$. Both codes are fully defined on the same set of qubits. The logical operators of each code are defined as

$$\mathscr{L}_{\text{old}} = \mathscr{C}(\mathscr{S}_{\text{old}}) \setminus \mathscr{S}_{\text{old}}, \qquad \mathscr{L}_{\text{new}} = \mathscr{C}(\mathscr{S}_{\text{new}}) \setminus \mathscr{S}_{\text{new}}.$$

We first need to define a joint subsystem code $\tilde{C}$ from these two codes. The general idea is that the gauge group of $\tilde{C}$, $\tilde{\mathscr{G}}$, is generated by both $\mathscr{S}_{\text{old}}$ and $\mathscr{S}_{\text{new}}$, so

$$\tilde{\mathscr{G}} = \langle \mathscr{S}_{\text{old}}, \mathscr{S}_{\text{new}} \rangle.$$

The center of $\tilde{\mathcal{G}}$, denoted $\tilde{\mathcal{S}}$, gives the stabilizer group of our subsystem code $\tilde{C}$. The elements in $\tilde{\mathcal{G}} \backslash \tilde{\mathcal{S}}$ are gauge operators, we denote them $\tilde{\mathcal{L}}_g$. Note that $\tilde{\mathcal{L}}_g$ does not necessarily contain the full algebra for a definite number of gauge qubits. There could be some Pauli operators in the gauge group whose anti-commuting counterparts are not. For example, this is the case in the lattice surgery merge, where $\mathcal{G}$ contains a representative of $\overline{Z}_1\overline{Z}_2$ but not any $\overline{X}_1$ or $\overline{X}_2$. In that case we extend $\tilde{\mathcal{G}}$ and $\tilde{\mathcal{L}}_g$ to $\mathcal{G}$ and $\mathcal{L}_g$ by adding the missing anti-commuting counterparts, fully defining the underlying subsystem code C. The gauge group $\mathcal{G}$ whose center is solely

$$\mathrm{Z}(\mathcal{G}) = \mathcal{S} = \mathcal{S}_{\mathrm{old}} \cap \mathcal{S}_{\mathrm{new}}.$$

Then, the logical operation that transforms C_{old} into C_{new} is realized by fixing some gauge operators in $\mathcal{L}_g = \mathcal{G} \backslash \mathcal{S}$. The group $\mathcal{M}_{\mathrm{fix}} \equiv \tilde{\mathcal{G}} \setminus \mathcal{S}_{\mathrm{old}}$ is the set of all fixed gauge operators. Once the subsystem code C is defined, one can verify the fault tolerance of this operation by checking three criteria:

1. **Code distance:** The distance of the subsystem code, C, must be large enough for the desired protection. Ideally it matches the distances of C_{old} and C_{new} so the degree of protection is not reduced during the deformation step.

2. **Error correction:** The error correction procedure follows that of the subsystem code C through the code deformation step.

3. **Gauge fixing:** To fix the gauge, one has to use operators exclusively from $\mathcal{L}_g$.

More specifically, criterion 2 means that to perform error correction, one has to reconstruct from the measurements of $\mathcal{S}_{\mathrm{new}}$ the syndrome given by $\mathcal{S}$. Importantly, criteria 2 and 3 demonstrate that the processes of error correction and that of gauge fixing are two separate processes with different functionality. Both processes require the application of Pauli operators (in hardware or in software) to make sure that stabilizer measurements are corrected to have outcome +1. The error correction process does this to correct for errors, while the gauge-fixing process does this to move from C_{old} to C_{new}. This description holds for one step of deformation, so that for each step in a sequence of deformations one has to examine the corresponding subsystem code C and its distance.

In addition, this discussion assumes that stabilizer measurements are perfect. When one considers noisy syndrome measurements, one needs to ensure that both the stabilizer outcomes and the state of the gauge qubits can be learned reliably. For 2D stabilizer codes such as the surface code this is simply done by repeating the measurements. To process this repeated measurement information for the surface code, one no longer uses the syndrome but the *difference syndrome*: the difference syndrome is marked as nontrivial (we say that a *defect* is present) only when the syndrome value changes from the previous round of measurement. This difference syndrome or defect gives information about both qubit errors as well as measurement errors. The construction of difference syndrome of a code deformation procedure is shown in Figure 3.5. The first round of measurement of $\mathcal{S}_{\mathrm{new}}$ at time T_d does not have a previous value to compare to in order to construct a difference syndrome. Therefore, one can only construct defects for $\mathcal{S}$. Immediately after this step, one can derive the difference syndrome of the full $\mathcal{S}_{\mathrm{new}}$,

placing defects accordingly. Using defects before and after T_d, one processes error information to infer the value of the gauge operators in $\mathcal{M}_{\text{fix}}$ at time T_g, thus fixing the gauge at T_g.

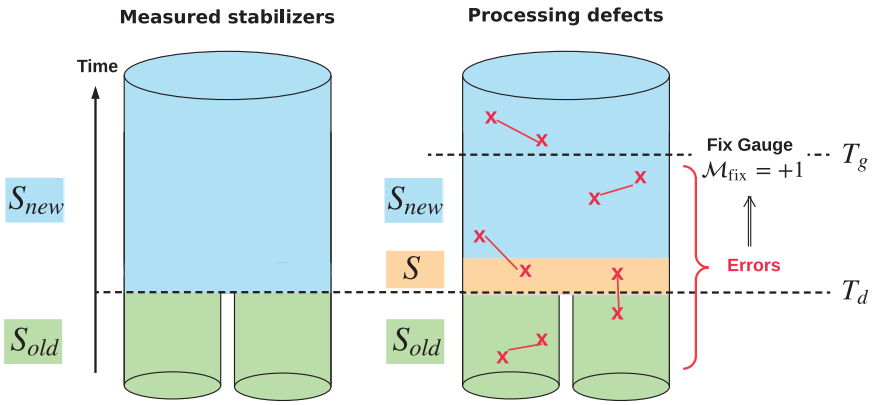
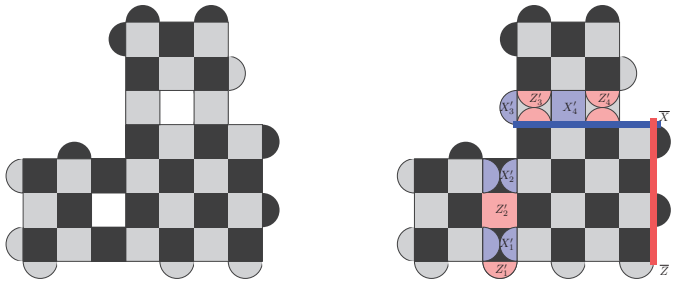


Figure 3.5: Schematic drawing of a code deformation procedure with repeated noisy measurements, with time increasing upwards. T_d designates the time step at which the code deformation (the switch from measuring the checks of $\mathcal{S}_{\text{old}}$ to those of $\mathcal{S}_{\text{new}}$) is performed. T_g is the time at which one is confident enough about the state of the gauge qubits, taking into account errors, to fix their states. This means that, after T_g, another logical computation can be performed.

In the remainder of this section, we apply this formalism to the code deformation and lattice surgery operations discussed earlier.

3.4.2. CODE DEFORMATION EXAMPLES

GROW OPERATIONS



(a) (b)

Figure 3.6: Description of the subsystem code, C, which holds during the first step of the grow operation depicted in Figure 3.1a and Figure 3.1b. (a) Generators for the stabilizer group, $\mathcal{S}$, of C. (b) Generators for the whole gauge group $\mathcal{G}$ of C. Highlighted in red and blue, respectively, are gauge operators, elements of $\mathcal{L}_g$, of Z-type and X-type, respectively. The logical operators, $\overline{X}, \overline{Z} \in \mathcal{L}_{\text{bare}}$, are also represented in brighter colors.

Gauge fixing, when applied to the growing operations of Figure 3.1 and Figure 3.2,

reveals an underlying subsystem code with a small number of widely-spaced holes and large boundaries, resulting in a high distance. The stabilizer group, $\mathscr{S}$, as well as the gauge operators, $\mathscr{L}_g$, for the subsystem code C which governs the deformation from Figure 3.1a to Figure 3.1b, are shown in Figure 3.6.

In all figures of this chapter, light blue and light red patches individually represent X-type and Z-type gauge operators, and bright blue and bright red qubit chains are $\overline{X}$ and $\overline{Z}$ operators respectively. The grow operation is changing the gauge from one in which the gauge operators not overlapping between the initially separate patches are fixed, denoted as $\{X_1', X_2', Z_3', Z_4'\}$ in Figure 3.6b, to one in which the overlapping ones are fixed, denoted as $\{Z_1', Z_2', X_3', X_4'\}$ in Figure 3.6b. The distance of C is still 5, matching the distance of the initial code.

Now consider what happens if we would go directly from Figure 3.1a to Figure 3.1c. The stabilizers and the gauge operators for this operation are shown in Figure 3.7. Similarly, one fixes the gauge going from separate patches to a single patch. The distance of the subsystem code for this operation is only 3. Indeed one of the minimum-weight dressed logical operators is the $\overline{Z}$ on the qubits in the green box in Figure 3.7b. That means that, in order to preserve the code distance, one should perform the intermediary step.

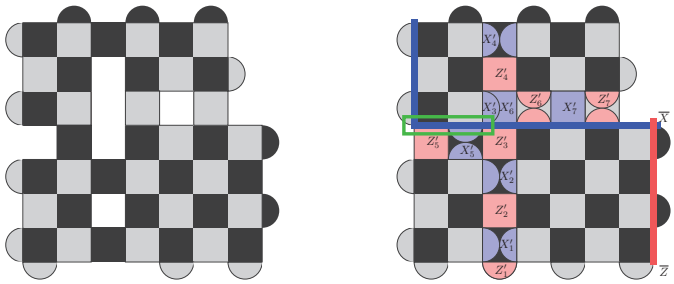


Figure 3.7: The operators of the subsystem code for the one-step grow operation from Figure 3.1a to Figure 3.1c, skipping Figure 3.1b: (a) The stabilizers which generate $\mathscr{S}$ and (b) the whole gauge group, $\mathscr{G}$, with highlighted gauge operators and logical operators.

THE MERGING AND SPLITTING OPERATIONS

In this section, we interpret the joint measurement of $\overline{ZZ}$ by lattice surgery in Figure 3.3b as gauge fixing. The stabilizer group $\mathscr{S}$ is generated by all the stabilizers in Figure 3.8a. The gauge operators, $\mathscr{L}_g$, of the gauge group are given by three representatives of the logical X of the top patch and the intermediary Z plaquettes that anti-commute with them. They are denoted as $\langle X_1', Z_1', X_2', Z_2', X_3', Z_3' \rangle$ in Figure 3.8b. Representatives of the bare logical operators, $\overline{X}, \overline{Z} \in \mathscr{L}_{\text{bare}}$, are the logical Z of the bottom patch and the logical X of the merged patch (joining the very top to the very bottom), see Figure 3.8b. The merge and split operations are realized by fixing some gauge operators of $\mathscr{L}_g$, resulting in new codes C_{merged} or C_{split}, respectively. Note that the weight of $\overline{X}$ of the subsystem

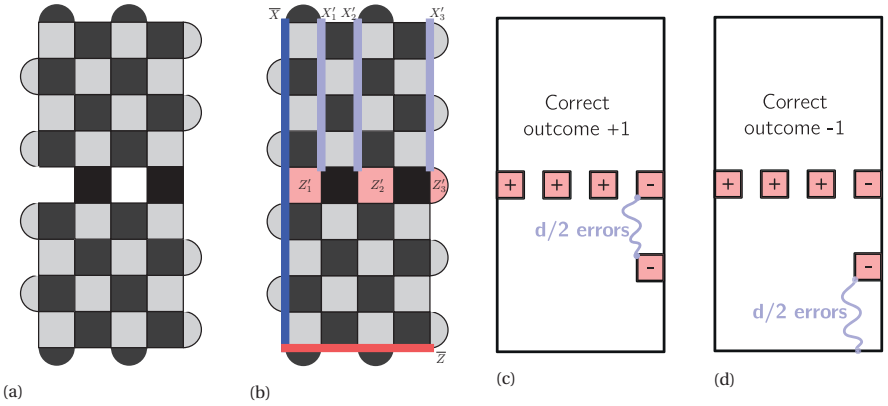


Figure 3.8: The operators of the subsystem code, C, for the joint measurement $\overline{ZZ}$. (a) The generators of stabilizer group $\mathscr{S}$. (b) The highlighted operators are either gauge operators in $\mathscr{L}_g$ or logical operators in $\mathscr{L}_{\text{bare}}$. We start in the gauge where the products $X_1'X_2'$ and $X_2'X_3'$ are fixed, and end in the gauge where Z_1', Z_2', and Z_3' are fixed. The distance of the subsystem code is 5, since one can construct a logical $\overline{X}$ with this weight by multiplying it with X gauge operators. (c) and (d) Two different scenarios with errors of weight $d/2$ with the same observed measurements.

code, C, is only d and not $2d$ which is the distance for X of the merged code. Indeed, by using the gauge operators like X_1' and stabilizers, one can construct a dressed logical X of weight d. Another way of seeing this is by realizing that one cannot distinguish between two errors of weight $d/2$ depicted in Figure 3.8c and Figure 3.8d. In the first one, the logical measurement outcome is -1 and there is a string of $d/2$ X-errors from the bottom to the middle of the bottom patch. In the second one the logical measurement outcome is $+1$ and there is a string of $d/2$ X-errors from the middle of the bottom patch and the middle (changing the observed logical measurement outcome to -1). Note also that when performing the splitting operation, one wants to correct the -1 outcomes for some of the intermediary X stabilizers. They are gauge operators equivalent to, say $X_1'X_2'$. They have to be corrected using the Z gauge operators, say Z_1' in this case. Otherwise one would introduce a logical Z error.

3.5. NUMERICS

To numerically evaluate the fault-tolerance of quantum computation on rotated planar surface codes, we simulate logical measurement, rotation, and logical CNOT, using the Gottesman-Knill formalism [25]. These simulations are carried out using two different error models, the *phenomenological* model and the *circuit-based* model. The phenomenological error model inserts independent X and Z errors on data qubits with equal probability p, and measurements output the wrong classical value with probability p. The circuit error model inserts errors with probability p after each operation of the error correction circuit as follows: each single-qubit gate is followed by a X, Y, or Z with probability $p/3$, each two-qubit gate is followed by an element of $\{I,X,Y,Z\}^{\otimes 2}\setminus\{II\}$ with probability $p/15$, and each measurement returns the wrong result with probability p. In this work, except when stated otherwise, the initial logical qubits are prepared without

errors when simulating these logical operations.

In Section 3.4.1, we have introduced how to construct defects (difference syndromes) for a code deformation step and how to process these defects to infer errors and fix gauge operators (Figure 3.5). For a realistic implementation of logical operations, a decoder will infer errors in a time window which may include T_d or T_g, by processing the defects within the window. This means the decoder should be able to match defects across time boundaries, e.g., the defects before and after code deformation time T_d. In addition, it needs to construct matching graphs with edges whose endpoints are on different lattices, e.g., defects of $\mathcal{S}_{\text{new}}$ may be matched to virtual defects beyond the past-time boundary T_d. However, such a decoder is difficult to implement. In our simulations, we insert perfect measurement rounds after blocks of d rounds of measurement (Figure 3.9) for ease of implementation, where d is the distance of the underlying subsystem code. A decoder using the minimum-weight perfect matching (MWPM) algorithm is used and its performance for a fault-tolerant memory operation, that is, d noisy QEC cycles followed by 1 noiseless cycle, is shown in Figure 3.10. For each operation (except for plain surgery), 10^5 (10^4) iterations were run per point and confidence intervals at 99.9% are plotted in the figures.

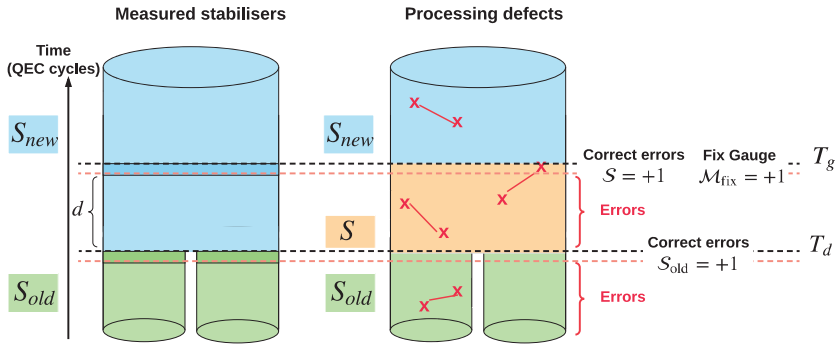


Figure 3.9: The simulated version of a code deformation procedure in Figure 3.5. A perfect round (a small time window from red to black dashed lines) is inserted after each block of noisy d rounds of stabilizer measurements. One processes the defects for $\mathcal{S}_{\text{old}}$ and corrects errors before the code deformation step T_d. Then the defects for $\mathcal{S}$ are constructed at time T_d to time T_g and the 'defects' for $\mathcal{M}_{\text{fix}}$ are constructed one round of measurement later. At time T_g, one processes error information to infer the value of the gauge operators and then fixes the gauge.

Single-qubit operations: Transversal operations (preparation, Pauli gates, measurement) are usually realized by performing qubit-wise physical operations. They are intrinsically fault-tolerant and their logical error rates will be only slightly higher than a logical identity gate (memory). Notably, a transversal $M_{\overline{Z}}$ ($M_{\overline{X}}$) measurement does not require quantum error correction cycles (i.e., $T_d = T_g$) since error syndromes of $Z(X)$-stabilizers can be reconstructed from the measurement outcomes of data qubits, this is also the case for the logical measurement step of plain surgery. For instance, one can measure all the data qubits in the Z basis to realize a $M_{\overline{Z}}$ on a planar surface code. After-

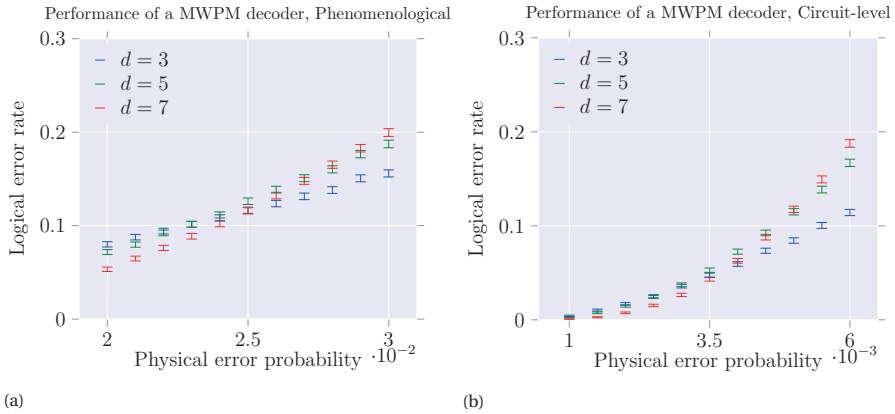


Figure 3.10: Numerical simulations of a fault-tolerant memory operation with the phenomenological error model near its threshold (~ 2.75% (a) and the circuit-level error model near its threshold (~ 0.5% (b).

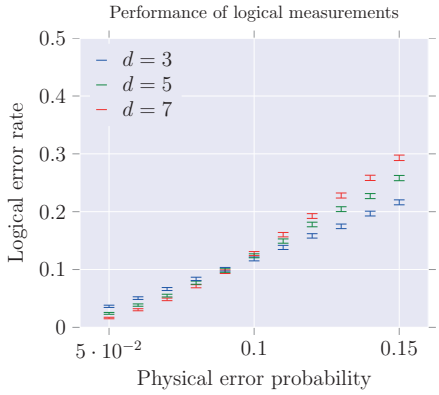


Figure 3.11: Numerical simulations of a transversal $M_{\overline{Z}}$ measurement near its threshold (~ 10%).

wards, one can compute the Z-syndromes by multiplying the outcomes of corresponding data qubits of each Z-stabilizer and then correct the X errors and deduce the value of $\overline{Z}$. The performance of a $M_{\overline{Z}}$ measurement for planar surface codes is shown in Figure 3.11. In this simulation, we first prepare a logical qubit in state $|\overline{0}\rangle$ without errors and then perform a $M_{\overline{Z}}$ measurement on it with physical measurement error probability p. We further numerically simulate the proposed rotating procedure (Figure 3.1) and show the results in Figure 3.12. For the phenomenological error model, the error threshold of a rotation is slightly lower than the threshold of quantum memory. For the circuit-level error model, its threshold is similar to that of quantum memory.

Two-qubit operations: We also simulate the measurement-based CNOT circuits in Figure 3.4a where the split operations of the first joint measurements are parallelized with the merge operations of the second joint measurements (see the decomposed circuits in Section 3.7.2). The overall error rates and the error thresholds for a CNOT gate

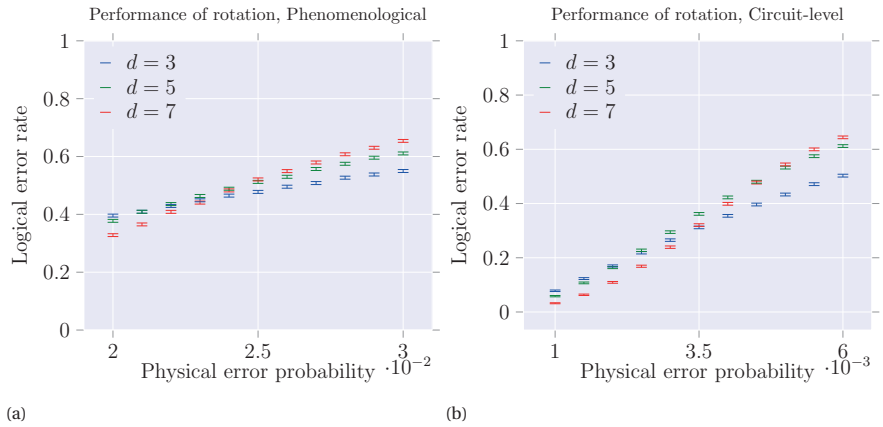


Figure 3.12: Numerical simulations of the rotation procedure in Figure 3.1 without a final flip operation. (a) and (b) The logical error rates of the rotation procedure with phenomenological error model (The error threshold is around ~ 2.5%) and circuit error model (The error threshold is around ~ 0.45%), respectively.

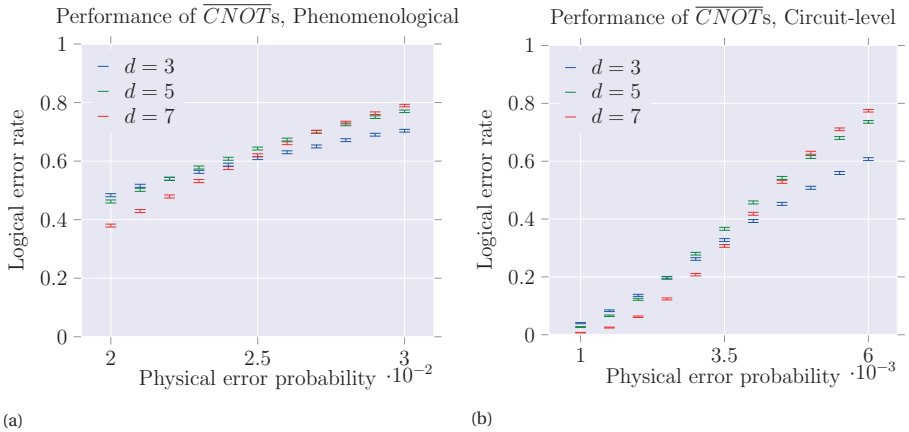


Figure 3.13: Numerical simulations of a measurement-based CNOT gate by lattice surgery (The top circuit in Figure 3.4a). (a) Total error rates for CNOT gates with the phenomenological error model near the threshold (~ 2.7%). (b) Total error rates for CNOT gates with the circuit-level error model near the threshold (~ 0.45%).

by lattice surgery are shown in Figure 3.13. For each error model, the error threshold of CNOT gates is similar to the threshold of quantum memory. Moreover, logical errors propagate through the measurement-based CNOT circuits, leading to a disparity of logical error rates on control and target qubits, which is demonstrated numerically in Section 3.7.2.

3.6. DISCUSSION & CONCLUSION

We have illustrated how to describe current measurement-based operations in 2D topological quantum computing using the gauge fixing technique. We have shown that, by

using the formalism of gauge fixing, the fault tolerance analysis of these code deformation and lattice surgery protocols is considerably simplified, their error correction and gauge fixing schemes also become clear. Furthermore, we numerically examined this method with examples on planar surface codes, including some well-known operations such as lattice-surgery-based CNOT gates and some novel protocols such as lattice rotation. Although this gauge fixing formalism does not provide direct guidelines on how to design code deformation protocols for a desired logical operation, it does provide an easy way to check the fault-tolerance of protocols and search for new ones via iterations of trial and error.

Moreover, this formalism applies not only to 2D topological codes, but more generally to any stabilizer code. In the general case (non-topological codes), the analysis of fault-tolerance in the presence of measurement errors becomes more involved, in particular with respect to how much repetition is really needed, see for example [65, 66]. We leave for future work how to obtain general and simple criteria for fault-tolerance.

3.7. APPENDIX

3.7.1. CODE CONVERSION AS GAUGE FIXING

To see the utility of gauge fixing for analyzing code conversion protocols, we consider two protocols for converting from the $[\![7, 1, 3]\!]$ Steane code to the $[\![15, 7, 3]\!]$ Reed-Muller code with six gauge Z operators fixed (see Figure 3.14 for the stabilizers and gauge operators that define these codes). The first, from Anderson et al [67], is based on the

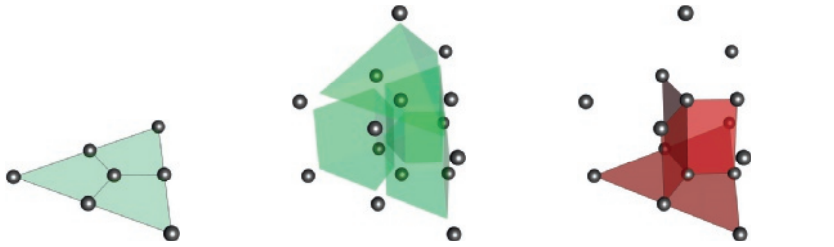


Figure 3.14: stabilizers of the Steane and Reed-Muller codes, and Z gauge operators of the Reed-Muller code. Red tinting on a face or volume indicates the presence of a Z operator on the vertices which make up that face or volume. For example, there are six Reed-Muller gauge operators of the form $Z^{\otimes 4}$, supported on the red-tinted quadrilaterals seen on the right. Green tinting indicates the presence of both an X and a Z stabilizer operator.

realization that the state $|\psi\rangle_{\text{Steane}} \otimes \frac{1}{\sqrt{2}}\left(|0\rangle_{\text{Steane}}|0\rangle + |1\rangle_{\text{Steane}}|1\rangle\right)$ is a code state of the Reed-Muller code with its horizontal X gauge logical operators fixed, see top-right of Figure 3.15. Conversion from the Steane code to the Reed-Muller code then involves fault-tolerantly preparing the eight-qubit ancilla state and fixing the three appropriate Z gauge operators. The state is always stabilized by the Reed-Muller stabilizers, whose eigenvalues can be reconstructed from the checks which are measured at every round, preserving the code distance and allowing error correction by syndrome decoding.

The second scheme, from Colladay and Mueller [68], is not based on gauge fixing,

and begins with the eight qubits needed for conversion initialized in the state $|0\rangle^{\otimes 8}$. This ensures that the initial checks anticommute with any potential X stabilizer supported on the final eight qubits, so that the only operators whose eigenvalues can be reconstructed from the measured operators are Z operators, preventing the correction of Z errors (see Figure 3.15 for a graphical comparison of these code conversion protocols). The difference in fault tolerance between these two protocols which accomplish the same task provides us with a good motive to incorporate subsystem codes into the analysis of code deformation and lattice surgery, considered in the main text.

Examining the Criterion 1 from Figure 3.4.1, one can see that the Anderson scheme has an underlying subsystem code with distance 3, whereas not having any X-stabilizers, the Colladay scheme has an underlying subsystem code with distance 1.

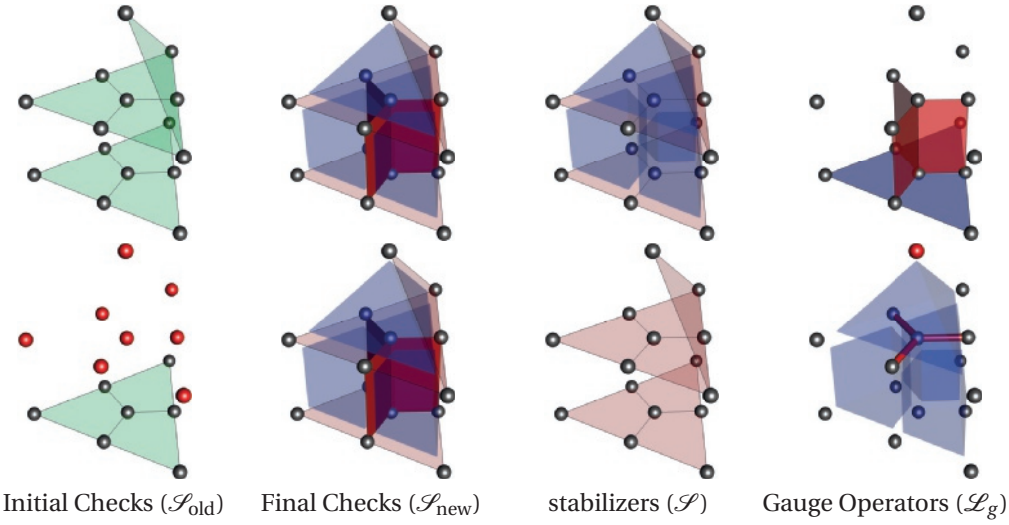


Figure 3.15: Comparison between Steane-to-Reed-Muller conversion schemes from [67] (top) and [68] (bottom). Red and green tinting match Figure 3.14, blue tinting indicates an X operator supported on the vertices of the tinted face or volume. Tinted vertices/edges indicate weight-one/two operators supported on the tinted vertex/edge. In the Anderson scheme, the subsystem code which applies during the code deformation is made explicit; it is the distance-three Reed-Muller code. The Colladay scheme, however, does not have any X operators in the relevant stabilizer, $\mathscr{S}$, so the distance of the relevant subsystem code is only 1, see Figure 3.4. Note: Gauge operators in the top right should also be present in the bottom right, they are not drawn here for clarity.

3.7.2. DISPARITY IN ERROR RATES OF CNOT GATES

A joint measurement is realized by performing a merge and a split operation in sequence. In our simulation, the circuits in Figure 3.4a are decomposed into the ones in Figure 3.16. Figure 3.17 shows that the rates of $\overline{X}/\overline{Z}$ errors on the control and target qubits are different for the rotated surface code with $d = 5$. This disparity can be explained using a toy model to account for propagation of logical errors through measurement-controlled corrections.

In this toy model, identity gates result in an $\overline{X}$ or $\overline{Z}$ error with probability p ($\overline{Y}$ errors are assumed to occur with probability $\sim p^2$, since the minimum-weight $\overline{Y}$ operator

has weight $2d-1$ in the surface code). The merge operations are modeled as ideal joint measurements, followed by an error of the form $\overline{X\otimes\mathbb{1}}$, $\overline{\mathbb{1}\otimes X}$, $\overline{Z\otimes\mathbb{1}}$, or $\overline{\mathbb{1}\otimes Z}$, each occurring with probability p, since these are the likeliest logical errors. If a logical Pauli error occurs, it propagates forward through the circuit, changing the measured eigenvalue for any measurement operator with which it anticommutes. For example, if an $\overline{X\otimes\mathbb{1}}$ error occurs after the M_{XX} operation in Figure 3.4a (in which the ancilla begins in the $|0\rangle$ state), the measured value b will be mapped to $1-b$, causing an $\overline{X}$ operator to be incorrectly applied to the target qubit at the end of the $\overline{\text{CNOT}}$. It is easy to confirm that there are 7 such first-order errors which result in an $\overline{X}$ error on the target qubit, 6 errors which result in a $\overline{Z}$ error on the control qubit, and 3 errors which result in the other logical errors shown in Figure 3.17a and Figure 3.17c (a similar analysis holds for the error rates shown in Figure 3.17b and Figure 3.17d). The biased logical error rates predicted by this simplified model are in good agreement with the logical error rates observed in simulation, shown in Figure 3.17. Preventing this bias from growing during the execution of a long algorithm, by appropriate selection of decomposition for CNOTs, is likely an important step in the design of high-performance fault-tolerant circuits for quantum computation.

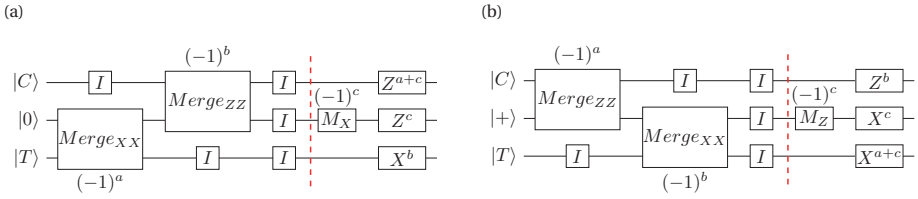


Figure 3.16: The decomposed circuits (a) and (b) of the top and bottom measurement-based CNOT circuits in Figure 3.4a.

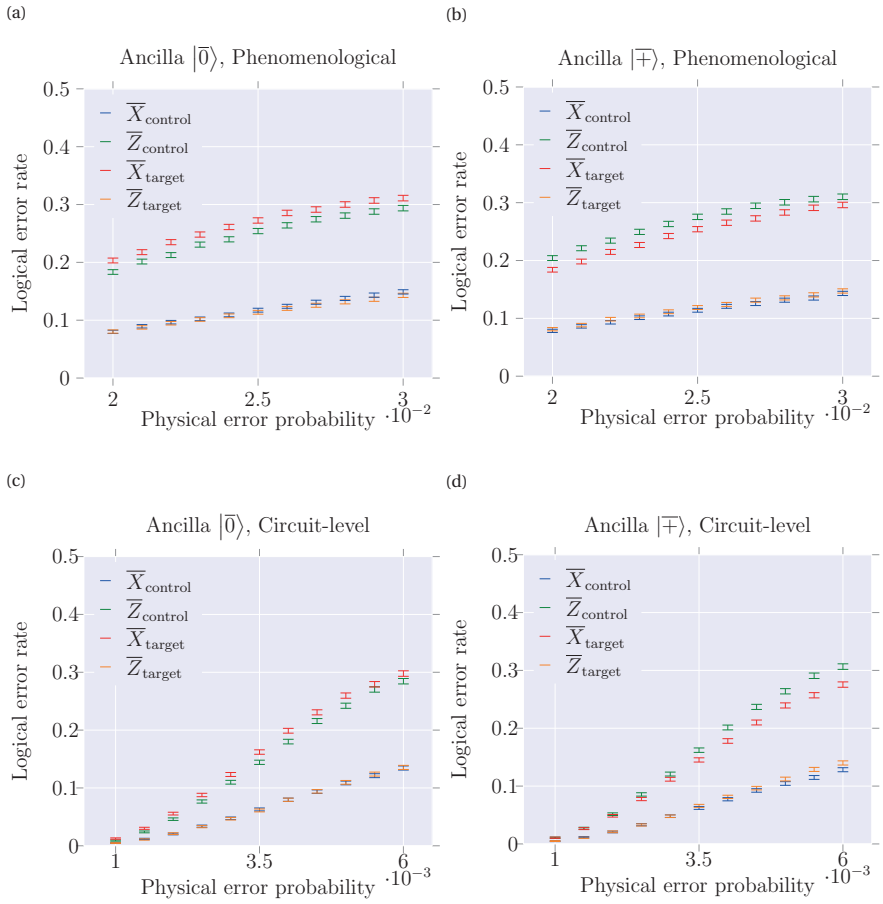


Figure 3.17: $\overline{X}$ and $\overline{Z}$ error rates on the control and target qubits for lattice-surgery-based CNOT operations at distance 5. (a) and (b) correspond to the phenomenological error model, (c) and (d) correspond to the circuit-based error model. The disparity in error rates is explained by error propagation through the measurement-based circuit implementing the CNOT.

4

Mapping of Lattice Surgery-based Quantum Circuits on Surface Code Architectures

Quantum error correction (QEC) and fault-tolerant (FT) mechanisms are essential for reliable quantum computing. However, QEC considerably increases the computation size and FT implementation has specific requirements on qubit layouts, causing both resource and time overhead. Reducing spatial-temporal costs becomes critical since it is beneficial to decrease the failure rate of quantum computation. To this purpose, scalable qubit plane architectures and efficient mapping passes including placement and routing of qubits as well as scheduling of operations are needed. This chapter proposes a full mapping process to execute lattice surgery-based quantum circuits on two surface code architectures, namely a checkerboard and a tile-based one. We show that the checkerboard architecture is twice as qubit-efficient but the tile-based one requires lower communication overhead in terms of both operation overhead (up to ~ 86%) and latency overhead (up to ~ 79%).

The contents of this chapter have been published in L. Lao, B. van Wee, I. Ashraf, J. van Someren, N. Khammassi, K. Bertels, C. G. Almudever, *Mapping Lattice Surgery-based Quantum Circuits onto Surface Code Architectures*, Quantum Science and Technology **4(1)**, 015005 (2019).

4.1. INTRODUCTION

BY exploiting superposition and entanglement, quantum computing can outperform classical computing while solving certain problems. For example, quantum computers can factor large numbers using Shor's algorithm with an exponential speedup over its best classical counterparts [1]. When adopting the circuit model as a computational model, algorithms can be described by quantum circuits consisting of qubits and gates. Such a circuit representation is hardware agnostic and assumes, for instance, that any arbitrary interaction between qubits is possible and both qubits and gates are reliable.

However, real quantum processors have specific constraints that must be complied to when executing a quantum algorithm, a procedure for mapping quantum circuits is therefore required. One of the main constraints in current quantum experimental platforms is the limited connectivity between qubits. A promising qubit structure that is being pursued for many quantum technologies like superconductors [16, 39] and quantum dots [17, 46], is a 2D grid architecture that only allows *nearest-neighbor* (NN) interactions. Other 2D qubit structures such as the quantum processors from IBM [14], Google [69], and Rigetti [45] have even more restrictive connectivity constraints. This means that non-neighboring or non-connected qubits need to be moved or routed to be adjacent for interacting -i.e. performing a two-qubit gate, resulting in an overhead in the number of operations as well as the execution time (latency) of the circuit.

Placing frequently interacting qubits close to each other combined with efficient routing techniques -e.g. shortest path- can help to reduce the movement overhead. In addition, exploiting available parallelism of operations will reduce the overall execution time of the circuit. Note that reducing the number of operations and the total circuit latency will be of benefit to decrease the failure rate of computation [70, 71]. Therefore, efficiently mapping quantum circuits on a specific qubit structure, including placement and routing of qubits and scheduling of operations, is necessary for reliable quantum computation. Many works have been done to map physical quantum circuits on different qubit structures. [72, 73, 74, 75, 76, 77, 78, 79] propose algorithms to map physical circuits on quantum processors with 2D NN structures. [80, 81, 82] and [83] respectively focus on IBM and Rigetti processors which both only support interactions on dedicated neighbors.

Moreover, quantum hardware is error prone, that is, the qubits loose their states (or decohere) extremely fast and quantum operations are faulty. For instance, superconducting qubits decohere in tens of microseconds [4] and quantum operations have error rates $\sim 0.1\%$ [3] compared to $\sim 10^{-15}$ for CMOS based devices. Therefore, *quantum error correction* (QEC) and *fault-tolerant* (FT) mechanisms are needed to protect quantum states from errors and make quantum computing FT. This is achieved by encoding a *logical* qubit into multiple error prone *physical* qubits and applying FT (logical) operations on such logical qubits [23]. However, QEC significantly increases the computation size up to four orders of magnitude [15]. Furthermore, this FT implementation may lead to more and/or different constraints on the encoded logical circuits, e.g., interaction restrictions between two logical qubits. Consequently, the mapping of fault-tolerant quantum circuits may become more difficult because it should consider both physical-level and logical-level constraints. In addition, it may require the definition of a virtual layer

called qubit plane architecture to provide scalable management of qubits and support fast execution of fault-tolerant operations.

Several papers [84, 85, 86, 87, 88, 89] have discussed how to map FT quantum circuits onto 2D quantum architectures based on concatenated codes such as the Steane code. However, not many papers focus on the surface code (SC) [90], currently one of the most promising QEC codes. [91, 92, 93] optimize quantum circuits based on defect surface codes in terms of geometrical volume defined by the product of # qubits and # gates (or time) of the circuit. [94] evaluates both planar and defect surface codes in terms of qubit resources and circuit latency. However, they assume two-qubit gates (CNOT) between two planar qubits (qubits are encoded in planar codes) can be performed transversally, which is an over-optimistic assumption given the limited connectivity in current quantum technologies. Fortunately, a technique called lattice surgery [18, 19] can be used to perform a two-qubit gate between two planar qubits in a 2D NN architecture. Nevertheless, the mapping of quantum circuits based on lattice surgery and the required qubit plane architecture have been hardly researched. [18] introduces a scalable qubit architecture for efficiently supporting lattice surgery-based two-qubit gates. [95] proves that the optimization of lattice surgery-based quantum circuits on its geometrical volume is NP-hard. This chapter will focus on the mapping of lattice surgery-based quantum circuits onto surface code qubit architectures. The contributions of this chapter are the following:

- We derive the logical-level constraints of the mapping process when the lattice surgery is used to perform FT operations on planar surface codes. We further provide the quantification of these logical operations, which are used for the mapping passes.

- Based on the qubit plane architecture presented in [18], we propose two different qubit architectures, namely a checkerboard architecture and a tile-based one, that support lattice surgery-based operations. For the tile-based architecture, we present an approach to fault-tolerantly swap tiles by lattice surgery, which is 3x faster than a standard SWAP operation by 3 consecutive logical CNOT gates. In addition, we also apply similar techniques to perform a FT CNOT gate between tiles where logical data qubits are not located in the required positions.

- We propose a full mapping procedure, including placement and routing of qubits and scheduling of operations, to map FT quantum circuits onto the two presented qubit architectures and evaluate these architectures on their communication overhead.

The chapter is organized as follows. Section 4.2 introduces the basics of FT quantum computing. We introduce two qubit plane architectures of interest in Section 4.3 followed by the proposed mapping passes in Section 4.4. The evaluation metrics and benchmarks are shown in Section 4.5. The experimental results are discussed in Section 4.6. Section 4.7 concludes.

4.2. FT QUANTUM COMPUTING

The surface code is one of the most promising QEC codes because of its high tolerance to errors (around 1%) and its simple 2D structure with only NN interactions as shown in Figure 2.3. In surface codes, there are two main ways of encoding a single logical qubit, using a *planar* [30] or a *defect* approach [96]. In the planar SC, a single lattice is used to encode one logical qubit. In the defect SC, a logical qubit is realized by creating defects in a lattice. However, the planar SC requires less physical qubits to encode one logical qubit for the same code distance. In the near-term implementation of quantum computing, qubits are scarce resources and current quantum technologies are pursuing a realization of planar SC quantum hardware [39]. This chapter therefore focuses on the planar surface code. Note that the FT implementation of the defect SC [15, 96, 59, 97] differs from the planar SC, leading to different implications on the mapping procedure.

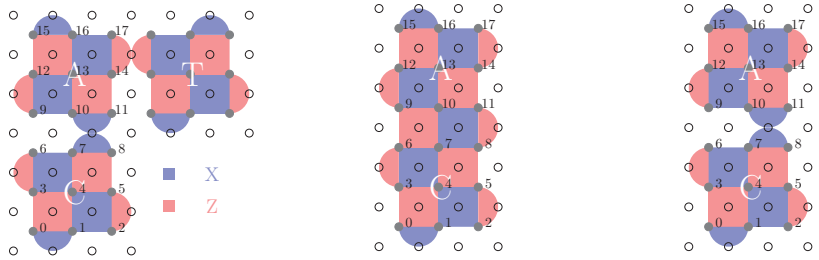


Figure 4.1: (a) Three planar SC-based logical qubits with $d = 3$. A 90-degree elbow-shaped qubit layout is required for implementing a lattice surgery-based CNOT gate between qubits 'C' and 'T'. 'C' is the control qubit, 'T' is the target qubit, and 'A' is the ancilla qubit in either $|0\rangle$ or $|+\rangle$ state. (b) The integrated lattice 'AC' and (c) the separated lattices 'A' and 'C', after merging and splitting 'A' and 'C', respectively.

4.2.1. FAULT-TOLERANT MECHANISMS

Figure 4.1a shows three logical qubits based on a distance-3 planar SC and they are labeled as 'A', 'T' and 'C', respectively. Each logical qubit consists of 17 physical qubits and has two types of boundaries, Z-boundaries and X-boundaries. For instance, in lattice 'A', the left and right boundaries are Z-type and the top and bottom boundaries are X-type. In the planar SC, initialization, measurement, Pauli gates, and H can be implemented transversally, i.e., applying bitwise physical operations on a subset of the data qubits, and then performing QEC cycles to detect and correct errors. The FT implementation of S and T gates in surface codes requires a non-deterministic procedure called magic state distillation. Since the S and T gates can be performed only if their corresponding magic states have been delivered, an online or dynamic scheduling and run-time routing may be required for efficient circuit execution [93]. In this chapter, we assume magic states have been prepared and properly allocated whenever S and T gates need to be performed.

In principle, a FT logical CNOT gate between two planar logical qubits can be performed transversally, i.e., applying pairwise physical CNOT gates to the data qubits in the two lattices. However, this transversal CNOT cannot be realized in current quan-

tum technologies which only allow NN interactions in 2D architectures. Alternatively, a measurement-based procedure [61] which is equivalent to a CNOT gate can be applied and its circuit representations are shown in Figure 3.4a. The joint measurement M_{XX} (M_{ZZ}) is realized by first merging two logical qubits and then splitting them, where their adjacent boundaries are Z-(X-)type boundaries. The outcomes of these measurements will determine whether the corresponding Pauli corrections should be applied (see Section 4.8.1 for more details).

The qubit layout for performing the measurement-based CNOT gate in the 2D NN architecture is shown in Figure 4.1a. The realization of the top circuit in Figure 3.4a is achieved as follows: 1) lattices 'A' and 'C' are merged and then split; 2) lattices 'A' and 'T' are merged and then split; and 3) measure 'A'. The merge and split operations are implemented by a technique called lattice surgery [18, 19]. For instance, the merge and split of lattice 'A' and 'C' are implemented by performing ESM on the integrated lattice (Figure 4.1b) and on the separated lattices (Figure 4.1c), respectively. In general, a surgery-based CNOT takes $4d+1$ SC cycles. It is worth mentioning that a split operation between qubits 'A' and 'C'('T') can happen simultaneously with a merge operation between qubits 'A' and 'T'('C'). Furthermore, a split operation between two qubits and a measurement on one of them can be performed in parallel. By exploiting the parallelism, the execution time in SC cycles can be reduced to $3d$.

4.2.2. IMPLICATIONS ON THE MAPPING PROBLEM

Based on the FT implementation of logical operations on planar surface codes, we derive the following constraints that must be taken into account by the mapping process as well as its implications.

Constraints: 1) The physical 2D NN interaction constraint is intrinsically satisfied by the construction of surface codes, thus the physical-level mapping becomes trivial; 2) A surgery-based CNOT gate requires that the qubits 'C' and 'T' together with the ancilla qubit 'A' are placed in particular neighboring positions, forming a 90-degree elbow-shaped layout.

Implications: 1) Logical qubits that need to interact and are not placed in such neighboring positions need to be moved, for instance by means of SWAP operations. The movement of qubits introduces overhead in terms of both qubit resources and execution time; 2) Therefore, in lattice surgery-based SC quantum computing, it is essential to predefine a qubit plane architecture for efficiently managing qubit resources and supporting communication between logical qubits; 3) In addition, operations for moving qubits should be defined; 4) It is necessary to initially place highly interacting logical qubits as close as possible and apply routing techniques to find the communication paths.

Based on the above observations, we will introduce two slightly different plane architectures and mapping passes for efficient execution of lattice surgery-based quantum circuits in the following sections.

4.3. QUBIT PLANE ARCHITECTURE

A qubit plane architecture is a virtual layer that organizes the qubits in different specialized and pre-defined areas such as communication, computation and storage [86, 88].

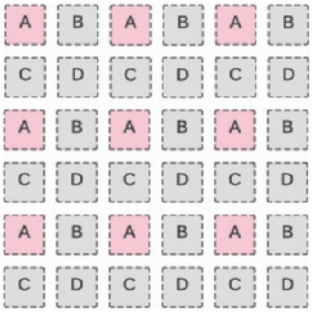
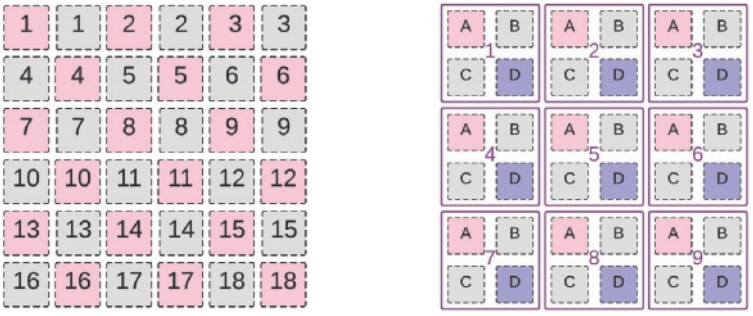


Figure 4.2: The qubit plane architecture proposed in [18] for lattice surgery-based planar surface codes, where each patch can hold one logical qubit shown in Figure 4.1a.



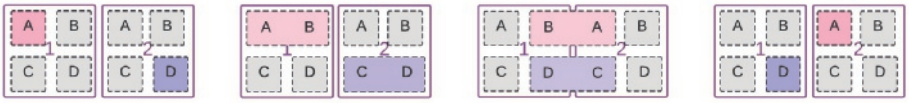
(a) (b)

Figure 4.3: (a) The checkerboard architecture (c-arch) and (b) the tile-based architecture (t-arch).

Qubit architectures should be able to manage qubit resources efficiently and provide fast execution of any quantum circuit.

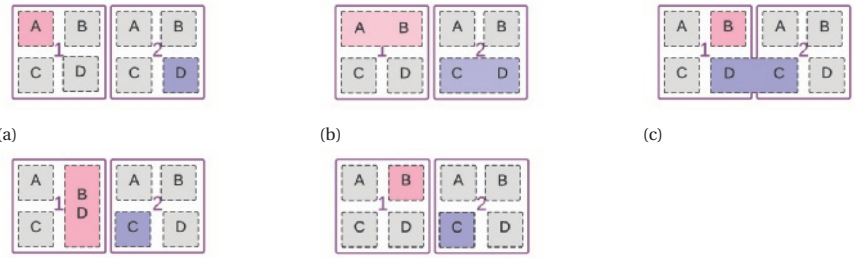
In [18], a layout that supports lattice surgery-based CNOT gates on planar surface codes is presented. As shown in Figure 4.2, it consists of several patches. The gray patches of the lattice are used for allowing qubits to perform CNOT operations, whereas the pink patches are used for holding logical data qubits. Then, only 1/4 of the available patches contains logical data qubits. Based on this layout, we propose two slightly different qubit plane architectures, the *tile-based* architecture (t-arch) and the *checkerboard* architecture (c-arch) as shown in Figure 4.3. The pink and blue patches are where logical data qubits containing information can be allocated (data patches), whereas the gray patches are assisting logical qubits (ancilla patches) that are used for performing logical CNOT gates and for communication. These two architectures differ in: i) the number of logical data qubits that can allocate, ii) the way movement operations are implemented, iii) the steps required for performing a CNOT between neighboring logical data qubits, and iv) the number of neighbors.

Logical data qubit allocation: In the checkerboard architecture, logical data qubits



(a) (b) (c) (d)

Figure 4.4: A t-SWAP between tiles 1 and 2. (a) Logical qubits are in patches A1 and D2. (b) Merge A1, B1 and D2, C2; (c) merge B1, A2 and C2,D1, and measure D2, A1; (d) measure B1, C2.



(a) (b) (c)

(d) (e)

Figure 4.5: A t-CNOT between tiles 1 and 2. (a) Control and target qubits are in patches A1 and D2 (b) Merge A1, B1 and D2, C2; (c) measure A1 and D2 and merge D1, C2; (d) merge B1,D1; (e) measure D1. The CNOT is performed in steps c), d) and e).

can be assigned to any of the pink patches, that is, 1/2 of the total patches are used to hold data qubits. In the tile-based architecture, a lined area consisting of 4 logical patches is defined as a basic computation tile and at most one logical data qubit can be allocated in each tile, that is, in either the pink or the blue patch. Then, only 1/4 of the total number of patches can be used for allocating logical data qubits.

Movement operations: One typical way to move physical qubits is through SWAP operations in which the state of the qubits is exchanged. Usually, a SWAP gate is implemented by applying 3 consecutive CNOT gates. The same principle can be applied for moving logical qubits. In this case a logical SWAP is realized by performing 3 consecutive lattice surgery-based CNOT gates, which is extremely time-consuming ($9d$ SC cycles). In the checkerboard architecture, we will use such a swap method called **c-SWAP** for moving logical qubits because of the limited number of ancilla patches. In the tile-based architecture, we propose to use a faster movement operation, which is analogous to the measurement-based procedure for CNOT gates, to swap data information between two horizontally or vertically adjacent tiles. This swap operation called **t-SWAP** only takes 1x logical CNOT gate time regardless of locations where data qubits are allocated inside the tiles -i.e. blue or pink patches. It is realized by 'moving' qubits to neighboring horizontal and vertical patches (see Section 4.8.2). Figure 4.4 shows an example of how to swap two logical data qubits placed in adjacent tiles by using the t-SWAP operation. Similarly, one can perform a t-SWAP between any other pair of patches in the horizontally or vertically adjacent tiles.

CNOT operations: As mentioned in Section 4.2, the control and target qubits need to be placed in patches that form a 90-degree elbow-shaped in order to perform a lattice

surgery-based CNOT. In the checkerboard architecture, two neighboring data patches are always in such 90-degree locations so that a lattice surgery-based CNOT gate can be directly performed between them. We called this operation **c-CNOT** and it is implemented by 3 steps, taking $3d$ SC cycles as described in the previous section. However, in the tile-based architecture, a CNOT operation called **t-CNOT** between two data qubits placed in horizontally, vertically, or diagonally adjacent tiles may need some pre-processing, depending on where data qubits are allocated. If the control and target logical qubits are already placed in patches forming a 90-degree shape, then one can perform the CNOT directly, e.g., patch D1 with patches A4, A2, A5. Otherwise, logical data qubits need to be moved to the required locations before performing the CNOT gate as shown in Figure 4.5.

Similarly, one can perform a t-CNOT between any other pair of patches in adjacent tiles. The t-CNOT with and without pre-processing takes $4d$ and $3d$ SC cycles, respectively. In the results section, we will assume that a t-CNOT always takes 4d SC cycles for simplicity.

Number of neighbors: In the checkerboard architecture one data patch can only interact with 4 adjacent data patches, e.g., the neighbors of patch 8 are $4, 5, 10, 11$ in Figure 4.3a. As mentioned in Section 4.2, a logical ancilla is required for performing a lattice surgery-based CNOT gate. To avoid ancilla conflicts when performing multiple logical CNOT gates simultaneously in the checkerboard, only the upper ancilla patch adjacent to the two interacting data patches can be used. For instance, ancilla 1 (2) will be used when performing a CNOT between data qubits 2 and 4 (2 and 5). In the tile-based architecture, one tile can interact with at most 8 neighbors, e.g., the neighbors of tile 5 are $1, 2, 3, 4, 6, 7, 8, 9$. However, logical CNOT gates between data qubits in tiles 1 and 5, and between data qubits in tiles 2 and 4 cannot be performed simultaneously because of ancilla conflicts. To avoid such conflicts for now, we only assume 6 neighbors per tile; we remove the right-top and left-bottom neighbors of each tile, e.g., remove tiles 3 and 7 from the neighbor list of tile 5.

In the next section, we will introduce the procedure for mapping lattice surgery-based quantum circuits onto both qubit architectures. We will then evaluate their communication overhead in terms of both operation overhead and latency overhead in Section 4.6 .

4.4. QUANTUM CIRCUIT MAPPING

The mapping of quantum circuits involves initial placement and routing of qubits and scheduling of operations. The need for QEC significantly enlarges the circuit size, which makes the mapping problem even more complex. For instance, in surface codes one logical qubit is encoded into $O(d^2)$ physical qubits and one lattice-surgery-based logical operation is implemented by $O(d^3)$ physical operations, where d is the code distance. Therefore, we propose to perform the mapping of quantum circuits before going to the physical implementation of logical qubits and operations. It means that each logical qubit is treated as one single unit, and each logical operation is regarded as one single instruction. Once the mapping is finished, logical operations need to be expanded into the corresponding physical operations. We use a library to translate each logical operation into pre-scheduled physical quantum operations (see Section 4.8.3). During the

Table 4.1: The execution time in SC cycles of different logical operations, d is the code distance.

	Init & MSMT	Pauli	H	S	T
# Cycles	1	1	$4d$	$14d$	$17d$
	c-CNOT	c-CNOT	t-CNOT	t-SWAP	
# Cycles	$3d$	$9d$	$4d$	$3d$	

translation, the address of underlying physical qubits corresponding to a logical qubit can be retrieved by maintaining a q-symbol table [22].

Table 4.1 depicts the execution time of different logical operations on planar surface codes expressed in SC cycles. It includes single-qubit operations as well as the two-qubit operations used in both qubit architectures presented previously. The execution time of different operations is used in the scheduling and routing passes. Furthermore, we will use these numbers for calculating the overall circuit latency in Section 5.

In order to illustrate the different steps in the mapping of quantum circuits, we will use the circuit in Figure 4.6a described by a quantum assembly language (QASM). This is the encoding circuit of the 7-qubit Steane code $[\![7, 1, 3]\!]$ and it can also be used to distill the magic states for S gates [15]. In this case, we assume each qubit is a logical qubit encoded by a distance-7 planar SC and each operation is a FT operation implemented by the techniques in Section 4.2 and Section 4.3.

4.4.1. SCHEDULING OPERATIONS

The objective of the scheduling problem is to minimize the total execution time (circuit latency) of quantum algorithms meanwhile keeping the correctness of the program semantics. Similar to instruction scheduling in classical processors, the correctness can be achieved by respecting the data dependency [98] between quantum operations. Analogous to classical computing, two kinds of data dependency can be defined for quantum computing: true dependency, which is the dependency between two single-qubit gates and between a single-qubit gate and a CNOT gate, and name dependency, which is the dependency between two CNOT gates which have the same control (or target) qubit.

We convert a QASM-described quantum circuit into a data flow-based weighted directed graph, which is called Quantum Operation Dependency Graph (QODG) and shown in Figure 4.6b. In this graph $G(V_G, E_G)$, each operation is denoted using a node v_i, and the data dependency arising from two consecutive operations on a same qubit, e.g., v_i followed by v_j, is represented using a directed edge $e(v_i, v_j)$. V_G and E_G are the node set and edge set of G, respectively. We also define E_G^1 and E_G^2 as the collection of edges that exhibits true and name dependency, respectively. S_{v_i} represents the starting time of operation v_i and T_{v_i} indicates its latency. The scheduling objective is to minimize the total circuit latency (Formula 4.1) while preserving the data dependency between operations (Formula 4.2).

$$\min \sup_{\forall v_i \in V_G} (S_{v_i} + T_{v_i}) \tag{4.1}$$

subject to

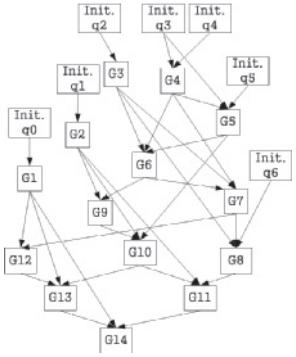
$$(S_{v_i} + T_{v_i}) \leqslant S_{v_j},\ \forall e(v_i, v_j) \in E_G \tag{4.2}$$

Note that two CNOT gates which share the same control or the same target qubit are commutable, meaning that they can be executed in any order except in parallel. This

```
.init q0, q1, q2,
 q3, q4, q5, q6

h q0   # (G1)
h q1   # (G2)
h q2   # (G3
cnot q3, q4   # (G4)
cnot q3, q5   # (G5)
cnot q2, q3   # (G6)
cnot q2, q4   # (G7)
cnot q2, q6   # (G8)
cnot q1, q3   # (G9)
cnot q1, q5   # (G10)
cnot q1, q6   # (G11)
cnot  q0, q4  # (G12)
cnot  q0, q5   # (G13)
cnot  q0, q6   # (G14)
```

(a)



(b)

```
prepz q0
h q0
qwait 19
{prepz q1 | prepz q2}
{h q1 | h q2}
qwait 5
{prepz q3 | prepz q4 | prepz q5}
{cnot q3,q5 | cnot q0,q4}
qwait 19
prepz q6
{cnot q3,q4 | cnot q2,q6 | cnot q1,q5}
qwait 20
{cnot q2,q4 | cnot q1,q3 | cnot q0,q6}
qwait 20
{cnot q2,q3 | cnot q1,q6 | cnot q0,q5}
```

(c)

Figure 4.6: The QASM description of the Steane ⟦7, 1, 3⟧ encoding circuit and its QODG. (a) The serial QASM representation; (b) The QODG; and (c) The scheduled parallel QASM representation. 'qwait' is an instruction specifies the waiting time until the next instruction can be issued.

commutation property has not been considered in previous works [89, 72, 73, 74, 75, 76, 84, 85, 86, 87, 88]. In this chapter, we take commuted CNOT gates into account and replace the optimization condition 4.2 with conditions 4.3 and 4.4:

$$S_{v_i} + T_{v_i} \leqslant S_{v_j}, \quad \forall e(v_i, v_j) \in E_G^1 \tag{4.3}$$

$$(S_{v_i} - S_{v_j}) \leqslant T_{v_j} \; or \; (S_{v_j} - S_{v_i}) \leqslant T_{v_i}, \; \forall e(v_i, v_j) \in E_G^2 \tag{4.4}$$

With respect to different dependencies, the scheduler will exploit parallelism and output the operation sequence with timing information, which is an as-soon-as-possible (ASAP) schedule. An as-late-as-possible (ALAP) schedule can be also easily implemented by scheduling operations in the reverse order (Figure 4.6c).

4.4.2. PLACING AND ROUTING QUBITS

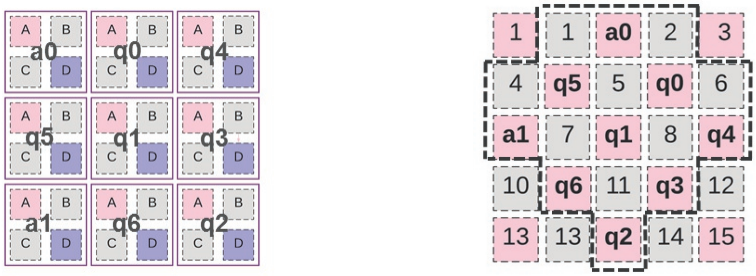
The QAP-model for initial placement of qubits: The goal of qubit placement is to find an optimal initial placement of the qubits that minimizes communication overhead. Similar to the placement approaches in [76, 86, 99], the initial placement problem is formulated as a quadratic assignment problem (QAP) with the communication overhead represented using the Manhattan distance:

$$\min\left(\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{n}\sum_{l=1}^{n} c_{ijkl} x_{ik} x_{jl}\right) \tag{4.5}$$

subject to

$$\sum_{i=1}^{m} x_{ik} = 1, \quad \forall k = 1, \cdots n \tag{4.6}$$

$$\sum_{k=1}^{n} x_{ik} = 1, \quad \forall i = 1, \cdots m \tag{4.7}$$



(a) (b)

Figure 4.7: The initial placements of the Steane ⟦7, 1, 3⟧ encoding circuit in (a) the tile-based architecture which has 3 × 3 tiles and (b) the checkerboard architecture which has 3 × 3 data patches in the dashed region (rotated by 45 degrees).

$$x_{ik} = \{0, 1\} \tag{4.8}$$

where $m(n)$ is the number of locations(qubits), $x_{ik(jl)} = 1$ or 0 indicates whether qubit $k(l)$ is assigned to location $i(j)$ or not, $c_{ijkl} = D_{ij}R_{kl}$ is the cost of separately assigning qubit k and l to locations i and j. D_{ij} is the Manhattan distance between locations i and j, and R_{kl} is the number of interactions between qubits k and l in the circuit. Constraints 4.6 and 4.7 ensure a one-to-one mapping from qubits to locations. A location is a tile in the tile-based architecture and a data patch in the checkerboard architecture. For instance, the initial placements of the Steane ⟦7, 1, 3⟧ encoding circuit in the $m = 3 \times 3$ tile-based architecture and the $m = 3 \times 3$ checkerboard architecture are shown in Figure 4.7.

In this chapter, the scheduling and QAP models are solved with integer linear programming (ILP). The scheduling uses the linearization method by [100], and the QAP uses the method proposed by [101]. ILP can only solve small-scale problems in reasonable time as the ones used in this chapter. Nevertheless, for near-term implementation in FT quantum computing, these numbers largely suffice. For large-scale circuits, one can either partition a large circuit into several smaller ones or apply heuristic algorithms to efficiently solve these mapping models [72, 73, 74, 76, 84, 85, 87].

The routing algorithm: The introduced two SC qubit architectures require routing of qubits, which involves finding communication paths and inserting the corresponding movement operations, for instance by means of the SWAP operations. An efficient routing should minimize the number of inserted movement operations as well as the increased latency. In this chapter, qubits are routed based on a sliding window (buffer) principle as shown in Algorithm 1. The algorithm will find a path for the first not NN instruction- i.e. CNOT operation in which qubits are not NN- inside the buffer. We adopted the breadth-first search (BFS) algorithm to find all possible shortest paths. Then, in order to select the communication path the algorithm looks back and forward. The look-back finds the maximum interleaving of movement instructions (SWAP) with previous instructions. The look-ahead will look how the positions of the qubits involved in a certain path is changed and how it affects future two-qubit operations; that is, we want to avoid to move away qubits that are already close to each other and need to interact in

Algorithm 1 Routing algorithm

Input: Defined qubit architecture and its size, initial placement, scheduled QASM-file
Output: Routed QASM-file

```
Define instruction buffer, B, length l = window size
Fill B with instructions from input-QASM
while B is not empty do
    # Check if an instruction (ins) is NN
    for ins in B[0 : l/2] do
        # If an instruction is not NN, start routing
        if ins is not NN then
            # Find different paths
            paths = all shortest paths for ins based on BFS
            # Look-back
            for p in paths do
                p.length = #cycles from in p − #cycles p can
                 interleave with instruction in B[0 : ins]
            # Look-ahead to other ins (o_i)
            for p in paths do
                Place qubits based on p
                p.length += ∑_{o_i∈B[ins:l]} shortest path
                 for o_i in #cycles
            Insert path with min. length and update placement
            Break for-loop
    Reschedule B[0 : ins]
    Write B[0] to output-QASM
    Fill B from input-QASM with qubit placement
```

the future. Once the path is selected, the instructions inside the buffer will be rescheduled using the ASAP strategy. Then the buffer will output routing instructions and will be fed with new ones. This process repeats until all CNOT gates can be performed in the pre-defined qubit architecture.

The results of routing the Steane $[\![7, 1, 3]\!]$ encoding circuit onto the tile-based and checkerboard architectures are shown in Figure 4.8 and Figure 4.9, respectively. The inputs of the routing process include 1) the pre-scheduled circuit using an ALAP approach in Figure 4.6c; 2) the initial placement in a pre-defined architecture in Figure 4.7. The routing process selects the communication path and inserts SWAP operations when two qubits for a coming CNOT gate are not neighbors and then the qubit layout is changed. Figures 4.8 and 4.9 show the final circuits with the intermediate qubit layouts after a full mapping procedure on the tile-based architecture and the checkerboard architecture respectively. Note that the operations inside each dashed block will be executed on the qubit layout marked in the same color and the current layout will be transformed into the next one after performing the inserted SWAP operation(s). Moreover, the final cir-

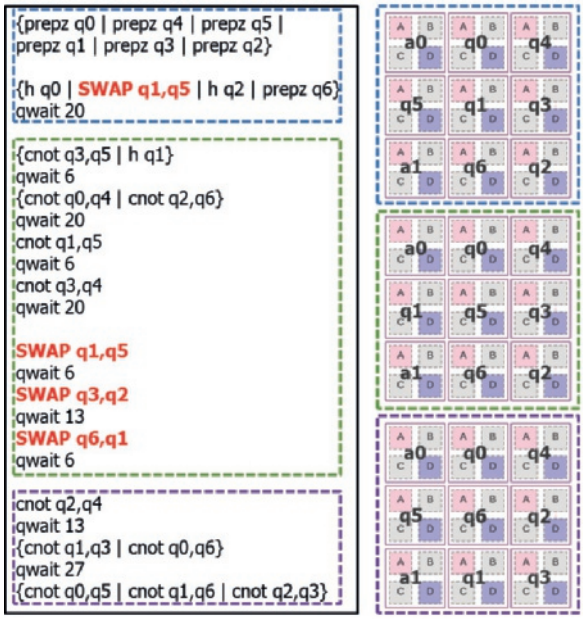


Figure 4.8: The final circuit and the intermediate qubit layouts after mapping the Steane ⟦7, 1, 3⟧ encoding circuit onto the tile-based architecture.

cuits after routing are totally different from the original circuit with an ALAP scheduling. This is because the operations inside each routing buffer has been rescheduled using an ASAP approach. For comparison purposes, a hand-optimized mapping of the Steane ⟦7, 1, 3⟧ encoding circuit on the 3×3 tile-based architecture is described in Section 4.8.4, including ALAP pre-scheduling, initial placement, and routing.

4.5. METRICS AND BENCHMARKS

In order to evaluate the impact of the mapping passes as well as the proposed qubit plane architectures we define the following metrics:

Qubit efficiency E_q: It is calculated as $E_q = \frac{\#Data}{\#AllQubits}$; where # AllQubits refers to the total number of logical qubits in a predefined qubit architecture for executing a quantum algorithm, including both logical data qubits and logical ancilla qubits, and # Data is the number of logical data qubits.

Circuit latency: It is the total execution time of a quantum algorithm in SC cycles. Even though reducing the circuit latency may have an overall negligible impact on the exponential performance improvement, it may be important for the algorithms with polynomial speedup. More importantly, shorter latency will also decrease the failure rate of the executed circuit.

Latency overhead: It is the percentage of latency used for moving qubits, and it is calculated as $\frac{L_R - L_S}{L_S}$; where L_R and L_S are the circuit latency with and without considering routing qubits, respectively.

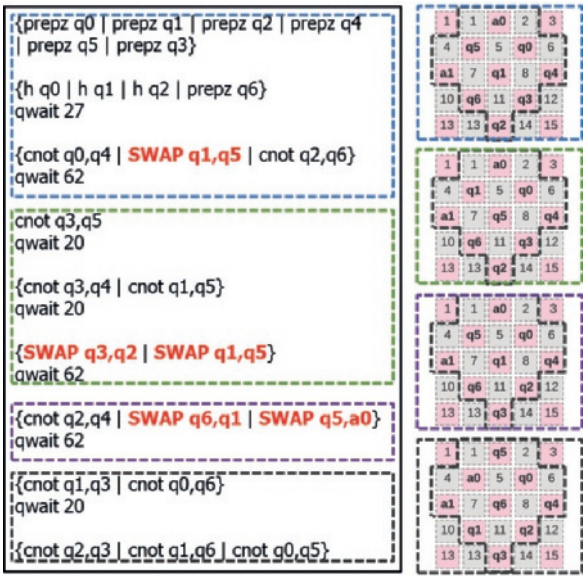


Figure 4.9: The final circuit and the intermediate qubit layouts after mapping the Steane ⟦7, 1, 3⟧ encoding circuit onto the checkerboard architecture.

Operation overhead: It is the percentage of inserted movement operations and it is calculated as $\frac{\#SWAPs}{\#Gates}$; where # Gates is the number of operations of the quantum algorithm which has not been routed (see Table 4.2) and # SWAPs is the total number of SWAP operations that are inserted for routing qubits. Reducing the number of operations for qubit communication helps to improve the computation fidelity.

Communication overhead: It is expressed in terms of both operation overhead and latency overhead.

The benchmarks used for this mapping evaluation are shown in Table 4.2 from Qlib [102] and RevLib [103]. These circuits are decomposed into ones which only contain the gates from the fault-tolerantly implementable universal set {Pauli, H, CNOT, S, T} on surface codes. We characterize these benchmarks in terms of percentage of CNOT gates $R_{cg} = \frac{\#CNOTs}{\#Gates}$, percentage of edges which have name dependency (E_G^2) in the QODG $R_{cd} = \frac{|E_G^2|}{|E_G|}$, and percentage of expensive $T, T^\dagger$ and $S, S^\dagger$ gates $R_{tsg} = \frac{(\#Ss+\#Ts+\#S^\dagger s+\#T^\dagger s)}{\#Gates}$. The first two benchmarks are encoding circuits of different QEC codes which are used for preparing magic states on SC [15]. Table 4.2 also shows the size ($R \times C$) of a qubit plane architecture, where R and C represent the number of data qubits in the x axis and y axis of the defined qubit plane architecture, respectively.

4.6. RESULTS

We map the benchmarks shown in Table 4.2 onto the two introduced qubit architectures using the proposed mapping passes. As shown in Table 4.1, the execution time of different operations is determined by the code distance d which is a tunable parameter of

Table 4.2: Quantum algorithm benchmarks

Benchmarks	# Qubits	# Gates	#CNOT	Rcg%	Rcd%	Rtsg%	Size
7-enc	7	21	12	52.38	42.55	0	3×3
15-enc	15	53	35	64.15	60.17	0	4×4
Adder0-5	16	306	126	41.18	26.1	48.0	4×4
Adder1-8	18	289	129	44.64	22.38	45.3	5×5
Adder1-16	34	577	257	44.54	22.16	44.9	6×6
Multiply4	21	1655	722	43.63	18.20	44.4	5×5
Shor15	11	4792	1788	37.31	21.03	48.4	4×3
sqrt7	15	7630	3089	40.48	6.41	43.72	4×4
sqrt8	12	3009	1314	43.67	4.63	43.50	4×3
ham7	7	320	149	35.63	5.67	41.56	3×3
hwb5	5	233	107	45.92	5.52	42.06	3×2
hwb6	6	1336	598	44.76	5.26	42.96	3×2
hwb7	7	6723	2952	43.91	4.64	43.62	3×3
rd73	10	230	104	45.22	4.61	42.61	4×3
rd84	15	343	154	44.90	4.63	42.86	4×4

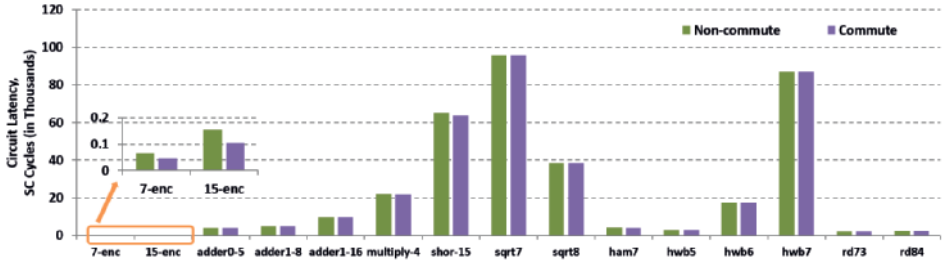


Figure 4.10: Comparison of the scheduling models with and without considering the commutation property ($d = 3$).

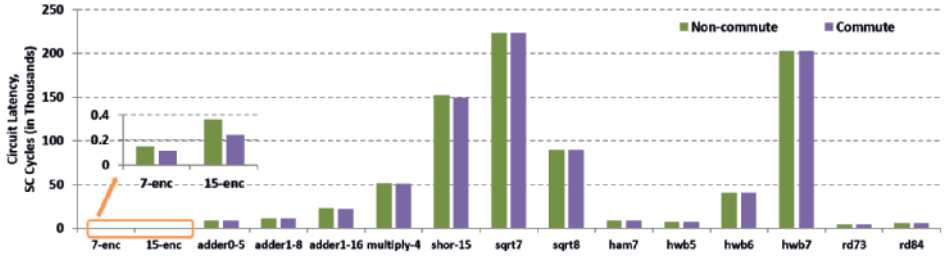


Figure 4.11: Comparison of the scheduling models with and without considering the commutation property ($d = 7$).

the mapping procedure. In this section, only the mapping results for a distance-3 and a distance-7 planar SC are presented, the results for other distances will be similar.

We first analyze the impact of the CNOT commutation property (Section 4.4.1) on the latency of scheduled quantum circuits. We only show the results for the ALAP scheduling as they are similar to the ASAP scheduling. Figure 4.10 (4.11) compares the proposed scheduling models for distance 3 (distance 7) with and without taking the commutation

property into account. For the encoding circuits, the scheduling considering commutation can significantly reduce the circuit latency, 28.1% (23.7%) for 7-enc and 34.4% (34.5%) for 15-enc, compared to the scheduling without considering commutation. This is because they have a high percentage of commutable CNOT gates (R_{cd}) meanwhile the percentage of expensive gates (R_{tsg}) is much lower (0). In contrast, for the other benchmarks the benefit of considering commutation is negligible, up to ~ 4% (~ 4%) for adder0-5.

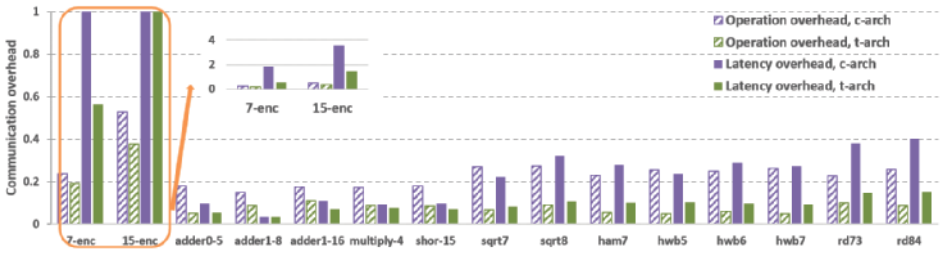


Figure 4.12: Comparison of mapping FT circuits onto different qubit architectures ($d = 3$). The latency overhead for the first two circuits are larger than 1 as shown in the sub-figure.

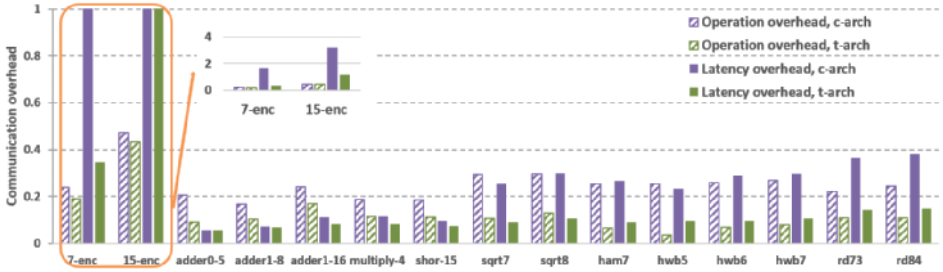


Figure 4.13: Comparison of mapping FT circuits onto different qubit architectures ($d = 7$).

Furthermore, we perform the full mapping procedure proposed in Section 4.4, including scheduling, placement and routing, on both the tile-based architecture (t-arch) and the checkerboard architecture (c-arch). The scheduling is implemented by the ALAP approach with considering commutation property. The initial placement is achieved by either the smart approach based on Manhattan distance or the naive method which places qubits in order. Note that the effect of initial placement is not always important [89], depending on the benchmarks (see Section 4.8.5). In this section, the best mapping result of the above two placement approaches for each benchmark is chosen.

Communication overhead: As mentioned previously, the mapping process results in an increase of the number of quantum operations (operation overhead) as well as in an increase in the circuit latency (latency overhead). We evaluate the communication overhead of mapping quantum circuits on different qubit plane architectures, namely the tile-based architecture (t-arch) and the checkerboard architecture (c-arch). Figures 4.12 and 4.13 show the comparison between t-arch and c-arch for distance 3 and 7 surface

codes, respectively. The mapping results for both distances are similar and the t-arch achieves less communication overhead because it has a higher number of nearest neighbors.

The operation overhead in the t-arch compared to the c-arch is reduced by 20.0% (7-enc) up to 81.4% (hwb5) for $d = 3$ and by 8.0% (15-enc) up to 86.4% (hwb5) for $d = 7$. The latency overhead when mapping on the t-arch shows a reduction of 3.2% (adder1-8), 16.3% (multiply-4) and up to 69.1% (7-enc) for distance 3. And 1.0% (adder1-8), 25.6% (shor-15) and up to 79.4% (7-enc) for distance 7. Note that this latency reduction is not only due to the less number of movement operations but also due to the use of much faster movements (t-SWAP) although the CNOT gates (t-CNOT) are slightly slower.

Qubit efficiency: As mentioned in Section 4.3, 1/4 and $\sim 1/2$ of the total number of patches are used for allocating logical data qubits in the tile-based architecture and the checkerboard architecture, respectively. Therefore, the qubit efficiency in the t-arch is $E_q = 1/4$ and the qubit efficiency in the c-arch is $E_q \approx 1/2$.

Based on the above observations, we can conclude that although the tile-based architecture is less qubit efficient than the checkerboard architecture, it can also substantially reduce the communication overhead in terms of operation overhead (up to $\sim$ 86%) and latency overhead (up to $\sim$ 79%). As we mentioned previously, decreasing the communication overhead helps to improve the computation fidelity. Therefore, one may have to compromise between qubit efficiency and communication overhead for the realization of quantum algorithms.

4.7. CONCLUSION

We have proposed two SC qubit plane architectures to efficiently support the execution of lattice surgery-based quantum circuits. We developed a full procedure for mapping small-scale quantum algorithms onto these two SC architectures. The experimental results show the following observations. First, the proposed scheduling considering the commutation property provides faster circuit execution than the scheduling without considering commutation. Secondly, the mapping procedure causes communication overhead in terms of both operation overhead and latency overhead. Moreover, the communication overhead highly depends on how qubits are organized and moved, that is, the qubit plane architectures. The tile-based architecture considerably decreases the number of movements and also supports faster execution compared to the checkerboard though it is less qubit-efficient.

As future work, we will focus on heuristic scheduling and placement algorithms as well as different routing techniques for large-scale quantum benchmarks. Furthermore, we will conssider the dynamics of quantum computation such as magic state distillation for S or T gates and qubit routing for performing 'neighboring' CNOT gates. Then we will investigate their implications on quantum circuit mapping. In addition, we will investigate different qubit architectures, for instance, an architecture with specialized communication channels for moving qubits and pre-defined regions for preparing magic states.

4.8. APPENDIX

4.8.1. LATTICE SURGERY-BASED CNOT

A CNOT is a gate applying on two qubits, the target qubit undergoes an X gate only if the control qubit is in $|1\rangle$. One way to validate a CNOT implementation is to check the transformation of logical X and Z operators using the Heisenberg representation [25] as follows:

$$CNOT^{\dagger}(X \otimes I)CNOT = X \otimes X \tag{4.9}$$

$$CNOT^{\dagger}(I \otimes X)CNOT = I \otimes X \tag{4.10}$$

$$CNOT^{\dagger}(Z \otimes I)CNOT = Z \otimes I \tag{4.11}$$

$$CNOT^{\dagger}(I \otimes Z)CNOT = Z \otimes Z \tag{4.12}$$

For instance, the CNOT gate transforms an X in the control qubit into the target qubit in Equation (4.9). We can verify the measurement-based procedure [61], which is described by the circuits in Figure 3.4a, by examining these transformations ((4.9)-(4.12)) as shown in Equations (4.13) and (4.14) respectively. These equations illustrate how different measurements transform stabilizers and logical operators. 'C', 'T', and 'A' represent the control, target, and ancillary qubit, respectively. 'S' and 'L' represent the stabilizers and logical operators, respectively. For example, after performing measurements M_{IXX} in (4.13), the stabilizer IZI is transformed into $(-1)^{M_{IXX}}IXX$ and the logical operator IIZ is transformed into IZZ. Equations (4.13) and (4.14) show that the measurement-based procedure does satisfy the transform relations in Equations (4.9)-(4.12) and it is thus equivalent to a CNOT.

	CAT						
S	IZI		$(-1)^{M_{IXX}}IXX$		$(-1)^{M_{ZZI}}ZZI$		$(-1)^{M_{IXI}}IXI$
L	XII		XII		$(-1)^{M_{IXX}}XXX$		$(-1)^{M_{IXX}+M_{IXI}}XIX$
	ZII	$\xrightarrow{M_{IXX}}$	ZII	$\xrightarrow{M_{ZZI}}$	ZII	$\xrightarrow{M_{IXI}}$	ZII
	IIX		IIX		IIX		IIX
	IIZ		IZZ		IZZ		$(-1)^{M_{ZZI}}ZIZ$

(4.13)

	CAT						
S	IXI		$(-1)^{M_{ZZI}}ZZI$		$(-1)^{M_{IXX}}IXX$		$(-1)^{M_{IZI}}IZI$
L	XII		XXI		XXI		$(-1)^{M_{IXX}}XIX$
	ZII	$\xrightarrow{M_{ZZI}}$	ZII	$\xrightarrow{M_{IXX}}$	ZII	$\xrightarrow{M_{IZI}}$	ZII
	IIX		IIX		IIX		IIX
	IIZ		IIZ		$(-1)^{M_{ZZI}}ZZZ$		$(-1)^{M_{ZZI}+M_{IZI}}ZIZ$

(4.14)

The joint measurement M_{XX} (M_{ZZ}) is realized by merge and split operations using lattice surgery [18, 19]. The basic operations of lattice surgery are to stop measuring some existing stabilizers and start to measure some new stabilizers. For example, the merge operation for M_{ZZ} on the qubits 'A' and 'C' in Figure 4.1a is performed by ceasing to measure X_7X_8 and $X_{10}X_{11}$, starting to measure $Z_6Z_7Z_9Z_{10}$, Z_8Z_{11} and $X_7X_8X_{10}X_{11}$, that is, performing d rounds of ESM on the merged lattice in Figure 4.1b. This means the two lattices 'A' and 'C' are integrated into one single lattice. Similarly, the split operation

is implemented by ceasing to measure $Z_6Z_7Z_9Z_{10}$, Z_8Z_{11} and $X_7X_8X_{10}X_{11}$, starting to measure stabilizer X_7X_8 and $X_{10}X_{11}$, that is, performing d rounds of ESM individually on each lattice 'A' and 'C' in Figure 4.1c. The splitting procedure divides the merged lattice back into two lattices. Afterwards, one needs to read out the outcome of each joint measurement for further logical Pauli corrections. The measurement result of M_{ZZ} is interpreted into 0 (1) if the number of '$-$' syndromes from the new stabilizers $Z_6Z_7Z_9Z_{10}$ and Z_8Z_{11} during the merge is even (odd).

4.8.2. LATTICE SURGERY-BASED MOVEMENT

The lattice surgery-based joint measurements can be used to 'move' logical qubits to other locations. As mentioned previously, the adjacent boundaries should be in both X- or Z-type when performing such a joint measurement. Assuming that the qubit patches in the same row (column) of the tile-based architecture in Figure 4.3b have $Z-(X$-)type adjacent boundaries, we introduce two basic movements: horizontal movement (Figure 4.14) and vertical movement (Figure 4.15). A logical state in A can be moved to its horizontally (vertically) adjacent position B (C) by first performing a joint measurement M_{XX} (M_{ZZ}) between A and B (C) followed by a Z (X) measurement on A. This horizontal (vertical) movements mimics the procedure in Equation (4.15) (Equation (4.16)), that is, the logical operators in patch A are transformed into patch B (C). It means that the logical state in A is moved to patch B (C). One can progressively move one logical state from one patch to the other by applying these horizontal movements and vertical movements as shown in Figure 4.16.

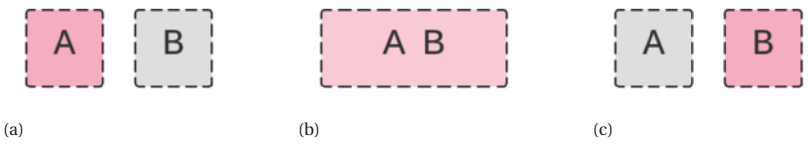


Figure 4.14: (a) Patch A is a logical qubit in state $|\psi\rangle$ and patch B is an ancilla in state $|0\rangle$. First perform the joint measurement M_{XX} realized by a merge (b) and a split (c), then perform the measurement M_Z on patch A, the state $|\psi\rangle$ is moved to patch B.

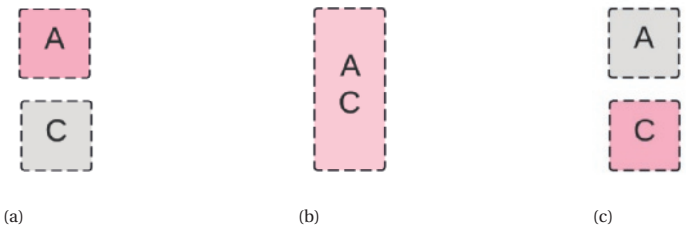


Figure 4.15: (a) Patch A is a logical qubit in state $|\psi\rangle$ and patch C is an ancilla in state $|+\rangle$. First perform the joint measurement M_{ZZ} realized by a merge (b) and a split (c), then perform the measurement M_X on patch A, the state $|\psi\rangle$ is moved to patch C.

	AB				
S	IZ		$(-1)^{M_{XX}} XX$		$(-1)^{M_{ZI}} ZI$
L	XI	M_{XX}	XI	M_{ZI}	$(-1)^{M_{XX}} IX$
	ZI		ZZ		$(-1)^{M_{ZI}} IZ$

(4.15)

	AC				
S	IX		$(-1)^{M_{ZZ}} ZZ$		$(-1)^{M_{XI}} XI$
L	XI	M_{ZZ}	XX	M_{XI}	$(-1)^{M_{XI}} IX$
	ZI		ZI		$(-1)^{M_{ZZ}} IZ$

(4.16)

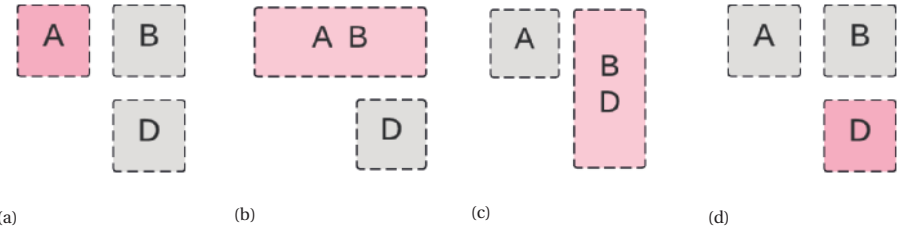


Figure 4.16: (a) Patch A is a logical qubit in state $|\psi\rangle$, patch B and D are ancillas in states $|0\rangle$ and $|+\rangle$, respectively. The state $|\psi\rangle$ is moved to patch D in $3d$ SC cycles as follows: (b) First perform the joint measurement M_{ZZ} between A and B; (c) and (d) then perform the joint measurement M_{XX} between B and D and finally perform the measurement M_X on patch A.

4.8.3. FT LIBRARY

After the logical-level mapping, the physical-level mapping becomes trivial for several reasons. First, there is no need to place and route physical qubits since surface codes intrinsically satisfy the 2D NN constraint. Secondly, as discussed in Section 4.2, each of the universal set of logical operations (preparation, measurement, Pauli, H, CNOT, S and T gates) on planar SC is implemented by a certain series of SC cycles.

As shown in Figure 4.17, each cycle is composed of two time slots, one blue slot for performing physical single-qubit gates and one gray slot for performing one round of ESM. Depending on the logical operation, a single-qubit gate such as Identity, Pauli gates or H gate needs to be performed during each blue slot. For instance, a logical X gate on the distance-3 planar surface code (Figure 2.3a) can be realized by one SC cycle, that is, first performing bit-wise physical X gates on qubits $D1, D2, D3$ (blue slot) and then performing 1 round of ESM (gray slot). Therefore, a library can be built to translate each logical operation into pre-scheduled physical quantum operations. Since the operations in a blue slot are bit-wise and performed in parallel, one only need to pre-schedule the operations of error syndrome measurement.

The ESM circuits for X and Z stabilizers are shown in Figure 2.3b. One full round of ESM on the distance-3 planar surface code (Figure 2.3a) is scheduled and performed as follows (in QASM):

{ prepz A2 | prepz A7 | prepz A5}
{ h A2 | h A7 | h A5 | prepz A1 | prepz A3 | prepz A6}
{ cnot A2, D5 | cnot A7, D9 | cnot A5, D7 | cnot D2, A1 | cnot D6, A3 | cnot D8, A6 | prepz A8 | prepz A4}

Figure 4.17: The decomposition of logical operations into SC cycles.

```
prepz q2
h q2
qwait 5
{prepz q3 | prepz q4}
cnot q3, q4
qwait 12
{prepz q0 | prepz q1}
{h q0 | h q1}
qwait 5
prepz q5
{cnot q3,q5 | cnot q2,q4}
qwait 19
prepz q6
{cnot q0,q4 | cnot q1,q5 | cnot q2,q6}
qwait 20
{cnot q2,q3 | cnot q0,q5 | cnot q1,q6}
qwait 20
{cnot q1,q3 | cnot q0,q6}
```

(a)

(b)

Figure 4.18: The hand-optimized (a) scheduling and (b) initial placement in a 3×3 tile-based architecture of the Steane $[\![7, 1, 3]\!]$ encoding circuit.

{ cnot A2, D2 | cnot A7, D6 | cnot A5, D4 | cnot D9, A8 | cnot D3, A3 | cnot D5, A6 | h A4}
{ cnot A2, D4 | cnot A7, D8 | cnot A4, D6 | cnot D1, A1 | cnot D5, A3 | cnot D7, A6 | h A5}
{ cnot A2, D1 | cnot A7, D5 | cnot A4, D3 | cnot D8, A8 | cnot D2, A3 | cnot D4, A6 | measure A1 | measure A5}
{ h A2 | h A4 | h A7 | measure A3 | measure A6 | measure A8}
{ measure A2 | measure A4 | measure A7}

However, a more realistic scheduling needs to consider the underlying hardware constraints such as the allowed primitive operations, their execution time, frequency multiplexing, etc. A scalable scheme for executing the ESM of surface code on superconducting qubits with NN coupling can be found in [39].

4.8.4. HAND-OPTIMIZED MAPPING EXAMPLE

In this section, we show a hand-optimized mapping of the Steane $[\![7, 1, 3]\!]$ encoding circuit in the 3×3 tile-based architecture. Using the Quantum Operation Dependency Graph shown in Figure 4.6b, instructions have been scheduled in an ALAP manner (Figure 4.18a). In addition, qubits have been placed in the lattice based on the number and frequency of interactions (Figure 4.18b). For the routing, one of the shortest paths is 'randomly' selected, resulting in the insertion of SWAPs shown in Figure 4.19. Compared with the proposed mapping procedure, the hand-optimized approach shows an increase of the communication overhead in terms of both latency overhead (15.9%) and operation overhead (25%).

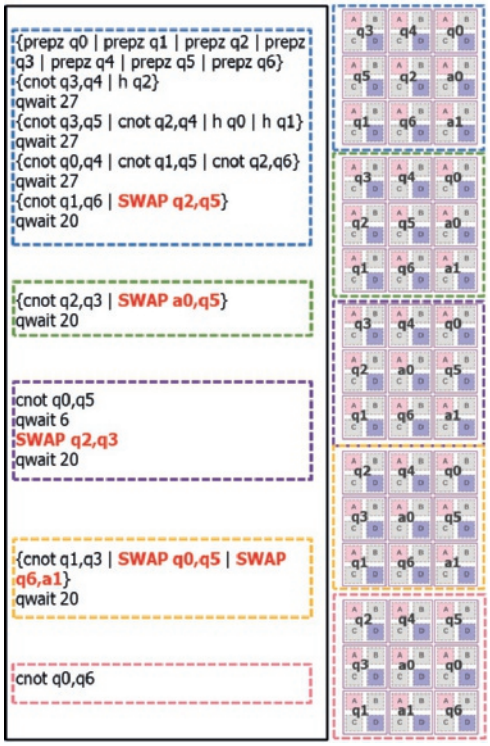


Figure 4.19: The final circuit and the intermediate qubit layouts after manually mapping the Steane ⟦7, 1, 3⟧ encoding circuit onto the tile-based architecture.

4.8.5. INITIAL PLACEMENTS

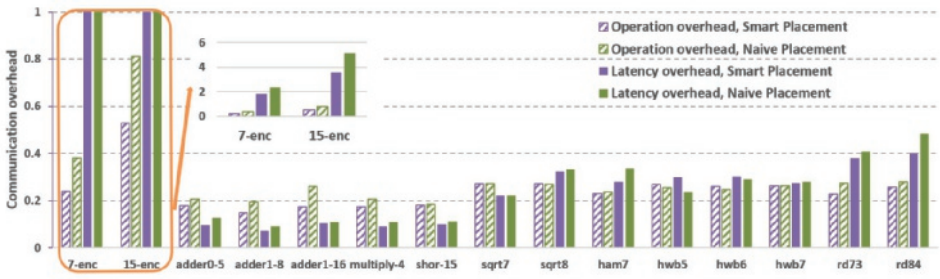


Figure 4.20: Comparison of mapping FT circuits with different initial placements on the checkerboard architecture ($d = 3$).

In this section, we examine how initial placement affects the mapping results.

Figures 4.20 and 4.21 show the comparison of the proposed smart placement based on Manhattan distance with a naive placement which locates qubits in order, where logical qubits are encoded by the distance-3 surface code. For some benchmarks, the

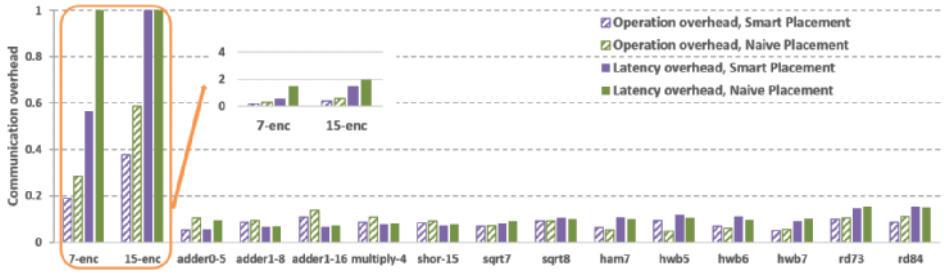


Figure 4.21: Comparison of mapping FT circuits with different initial placements on the tile-based architecture ($d = 3$).

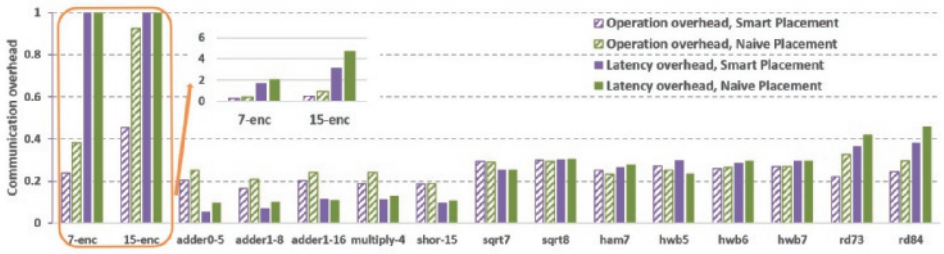


Figure 4.22: Comparison of mapping FT circuits with different initial placements on the checkerboard architecture ($d = 7$).

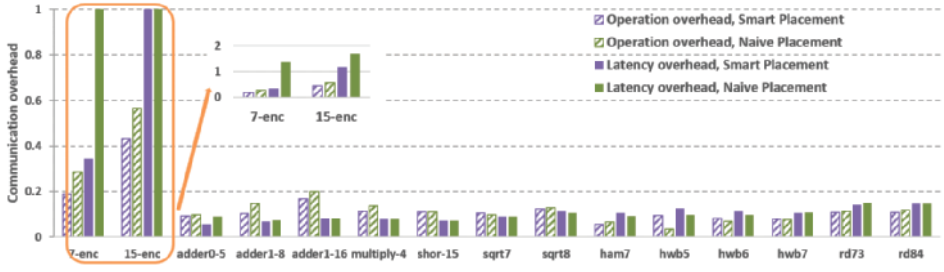


Figure 4.23: Comparison of mapping FT circuits with different initial placements on the tile-based architecture ($d = 7$).

use of the smart initial placement effectively decreases the operation overhead on both the c-arch and the t-arch, from ~ 8.3% up to 37.5% (rd84, adder0-5, multiply-4, rd73, adder1-8, adder1-16, 15-enc) and from ~ 7.4% up to 50% (adder1-8, hwb7, shor-15, multiply-4, rd84, adder1-16, 7-enc, 15-enc), respectively. Furthermore, the smart placement approach reduces the latency overhead of the c-arch and t-arch by 7.0% to 31.1% (rd73, shor-15, rd84, multiply-4, ham7, adder1-8, 7-enc, adder0-5, 15-enc) and by 7.4% to 62.3% (shor-15, adder1-16, hwb7, sqrt7, 15-enc, adder0-5, 7-enc), respectively. However, for other benchmarks, the smart placements provide marginal reductions or even increases in communication overhead on both qubit architectures. This is because the

position of the qubits will change after each SWAP operation, and the possible benefit of the smart initial placement will progressively disappear as the circuit execution advances.

Figures 4.22 and 4.23 show similar results for distance-7 surface code. For some benchmarks, the use of smart initial placements can effectively decrease the communication overhead compared to naive placements. The smart initial placement decreases the operation overhead on the c-arch and the t-arch, from ~ 15.8% up to 51.0% (adder1-16, rd84, adder0-5, adder1-8, 7-enc, 15-enc) and from ~ 6.7% up to 33.3% (adder0-5, rd84, ham7, adder1-16, multiply-4, 15-enc, adder1-8, 7-enc), respectively. Moreover, the smart placement approach reduces the latency overhead of the c-arch and t-arch by 10.2% to 31.1% (shor-15, multiply-4, rd73, rd84, 7-enc, adder1-8, 15-enc, adder0-5) and by 5.7% to 74.7% (rd73, adder1-8, 15-enc, adder0-5, 7-enc), respectively. However, for other benchmarks, the benefits from smart initial placements disappear on both qubit architectures.

5

A Control Microarchitecture for Fault-Tolerant Quantum Computing

Quantum computers can solve problems that are inefficiently solved by best known classical algorithms, such as integer factorization. A fully programmable quantum computer requires a quantum control microarchitecture that connects the quantum software and hardware. Previous research has proposed a Quantum Instruction Set Architecture (QISA) and a quantum control microarchitecture, which targets Noisy Intermediate-Scale Quantum (NISQ) devices without fault-tolerance. However, fault-tolerant (FT) quantum computing requires FT implementation of logical operations, and repeated quantum error correction, possibly at runtime. Though highly patterned, the amount of required (physical) operations to perform logical operations is ample, which cannot be well executed by existing quantum control microarchitectures.

In this chapter, we propose a control microarchitecture that can efficiently support fault-tolerant quantum computing based on the rotated planar surface code with logical operations implemented by lattice surgery. It highlights a two-level address mechanism which enables a clean compilation model for a large number of qubits, and microarchitectural support for quantum error correction at runtime, which can significantly reduce the quantum program codesize and present better scalability.

The contents of this chapter have been published in X. Fu*, L. Lao*, K. Bertels, C. G. Almudever, *A Control Microarchitecture for Fault-tolerant Quantum Computing*, Microprocessors and Microsystems **70**, 21-30 (2019).
*Those authors contribute equally to this paper.

5.1. INTRODUCTION

Quantum computers are promising because of their capability of solving problems that are currently inefficiently solved by classical computers, such as integer factorization [1] and quantum chemistry simulation [104, 105]. A fully programmable quantum computer based on the circuit model [23] requires a seamless collaboration between quantum software and quantum hardware with a quantum instruction set architecture (QISA) and quantum control microarchitecture serving as an interface [106] .

Due to the short coherence time of qubits and the erroneous quantum operations, fault-tolerant quantum computing (FTQC) based on quantum error correction (QEC) is essential to implement large-scale quantum algorithms. The basic idea of QEC is to encode quantum information into a *logical qubit* using a group of *physical qubits* according to some encoding scheme called quantum error correction code (QECC). To achieve fault-tolerance, it requires periodically detecting and (if necessary) correcting possible quantum errors through a highly patterned process called error syndrome measurement (ESM). In addition, quantum operations on such logical qubits should be implemented by a series of physical operations in such a way that individual errors of physical operations will not ruin the information stored in the logical qubits. As a consequence, FTQC dramatically increases the number of required physical qubits and the number of physical operations.

However, quantum control microarchitectures and QISAs proposed by recent research [106, 107] mainly target Noisy Intermediate-Scale Quantum (NISQ) devices with around fifty to hundreds of qubits [21], where quantum error correction is not applied. The QISA required by FTQC can be different to that required by NISQ technology because of the following reasons.

- Quantum algorithms targeting NISQ technology directly operate on individual physical qubits without QEC. In contrast, quantum algorithms operate on logical qubits in FTQC. The microarchitecture should support not only logical operations but also individual physical operations, which are required to implement some logical operations such as initialization [15].

- QEC introduces more complex classical computational tasks at runtime, such as quantum error decoding and error tracking using Pauli frame [108, 109], which require the support of new instructions in the QISA and new blocks at the control microarchitecture.

- The quantum error correction process requires repeated physical operations on qubits. It significantly increases the number of quantum operations on qubits per unit time and aggravates the quantum operation issue rate problem [106] confronting the control microarchitecture.

It is an open challenge to develop a scalable and flexible control microarchitecture that can satisfy the requirement of quantum error correction and fault-tolerant logical operations. This chapter envisions a Fault-Tolerant Quantum MicroArchitecture, FTQuMA, for the rotated planar surface code with logical operations implemented by lattice surgery [19]. The main contributions of this chapter are the following:

- We introduce the concept *virtual memory* into quantum computing with microarchitectural support, which contributes to a clean compilation model independent of the actual physical addresses of qubits that can vary from device to device;

- We propose a scheme to support quantum error detection and correction at the microarchitecture level, which can enable flexible planar-surface-code-based fault-tolerant logical operations implemented by lattice surgery;

- We propose a hardware mechanism that substantially reduces the codesize of the executable to enable efficient execution of quantum instructions.

5.2. FAULT-TOLERANT QUANTUM COMPUTING

5.2.1. QUANTUM ERROR CORRECTION

Qubits are fragile and quantum operations are erroneous. QEC can protect quantum states against errors by encoding one logical qubit into several physical qubits, which are called *data qubits*. Logical operations can be implemented by a series of physical operations on the physical qubits in such a way so that individual errors can be detected and corrected to achieve fault-tolerance. Non-destructive error syndrome measurement is periodically performed with the assistance of ancillary qubits, called *ancilla qubits*, to discretize quantum errors and extract the error syndromes. Afterwards, quantum error decoding is applied to find the likely errors based on the observed syndromes [30, 110, 111, 36, 112]. The capability of a QECC to detect and correct errors is characterized by the *distance d*, which is defined as the minimum number of physical operations required to implement a logical operation.

The surface code [28, 15] is a two-dimensional (2D) topological stabilizer code of which ESM is realized by measuring low-weight stabilizers, that is applicable on near-term quantum devices with limited connectivity. Moreover, the surface code has high tolerance to errors with an error threshold around 1%, which can be achieved by several quantum technologies such as superconducting qubits. This chapter focuses on the rotated planar surface code and investigates its implications on a quantum microarchitecture. The qubit layout of a distance-3 rotated planar surface code is shown in Figure 2.3a. The circuits for performing X- and Z-stabilizer measurements are shown in Figure 2.3b. It is worth mentioning that these two-qubit gates need to be performed in a specific order to ensure fault tolerance, i.e., an 'S'-shape order and an 'N'-shape order for X- and Z-stabilizers, respectively [113]. Both circuits end with a measurement on the ancilla qubit. The measurement results (binary values), or error syndromes, are then forwarded to a classical module, where decoding algorithms such as minimum weight perfect matching [31, 32, 33] are used to identify the possible errors with high likelihood and propose the corresponding corrections. Rather than physically performing these corrections, which may introduce more errors to the quantum system, classical control logic can track these errors using a technique called *Pauli frame* [108] that can be implemented in the microarchitecture [109].

5.2.2. FAULT-TOLERANT LOGICAL OPERATIONS

A circuit-model-based universal quantum computer requires at least a universal set of logical operations [23]. For the rotated planar surface code, such an implementable set includes:

1. The preparation, measurement, and Pauli gates, which can be implemented transversally;
2. The H gate and CNOT gate, which can be implemented using lattice surgery;
3. The S gate and T gate, requiring magic state distillation.

Note that, other surface code schemes such as twists [114, 58] and defects [15, 56] have a different implementation of logical operations.

TRANSVERSAL LOGICAL OPERATIONS

In planar surface codes, logical Pauli gates, preparation ($|0\rangle$ or $|+\rangle$) and measurement (in the X or Z basis) can be implemented transversally [15], i.e. applying bitwise operations to data qubits of the logical qubit followed by 0 - d rounds of ESM to decode errors. For example, the logical X and Z on a distance-3 surface code (e.g., lattice 'C' in Figure 2.3a) can be realized by performing $X_{D1}X_{D2}X_{D3}$ and $Z_{D1}Z_{D4}Z_{D7}$ followed by 3 rounds of ESM, respectively. Note that in This chapter, unless otherwise specified, logical preparation refers to the preparation of state $|0\rangle$ or $|+\rangle$, and logical measurement means measuring in the X or Z basis.

LOGICAL CNOT & H GATES

In order to perform logical CNOT gates in a 2D layout with only nearest-neighbor (NN) interactions, Gottesman [61] proposed a measurement-based procedure which can be implemented by the circuit shown in Figure 3.4a. The qubit layout for performing such a logical CNOT gate on a distance-3 planar surface code is shown in Figure 4.1a. The joint measurement M_{XX} or M_{ZZ} is realized by first merging the lattices of these two logical qubits and then splitting them using a technique called *lattice surgery* [18, 19]. For example, the merge operation required by M_{ZZ} is realized by performing d rounds of ESM on the merged lattice of the two logical qubits as shown in Figure 4.1b. After the merge, the two logical qubits 'A' and 'C' are integrated into one single lattice. Similarly, the splitting operation is implemented by performing d rounds of ESM individually on each lattice 'A' and 'C' in Figure 4.1c. The splitting procedure divides the merged lattice back into two lattices. After the merging and splitting operation, the joint measurement result can be retrieved by multiplying the measurement results of some new stabilizers introduced by the merged lattice after correcting all errors, which will be used for further logical Pauli corrections (Figure 3.4a). For instance, the measurement result of M_{ZZ} is the multiplication of the outcomes of the two newly-introduced Z stabilizers (framed) in Figure 4.1b. In total, $\sim 3d$ rounds of ESM are performed to realize a lattice-surgery-based CNOT gate.

Note that the error decoding for lattice-surgery-based operations may differ from the decoding for the standard error correction cycles (see [64] for more technical details). For

the merged lattice in Figure 4.1b, detecting Z errors requires decoding over the measurement outcomes of all the X-stabilizers in the merged lattice, but detecting X errors only requires decoding over the measurement outcomes of all the Z-stabilizers in the original two separate lattices.

In principle, the logical H gates on a planar surface code can be implemented transversally. Afterwards, the X- and Z-stabilizers will be exchanged. The location of X- and Z-stabilizers of each lattice – which property we name the *base ancilla type* (see Figure 5.3) – should be kept track of to generate the correct ESM circuits. A logical CNOT can only be performed between two lattices whose base ancilla types are the same. For example, the logical qubit 'C' and 'T' in Figure 4.1a have the same base ancilla type so that a logical CNOT gate can be directly performed between them. If the base ancilla types are different, one of the lattices should be rotated by 90° to align the base ancilla types before a logical CNOT gate. Previous works have proposed several lattice-surgery-based approaches to rotate a planar surface code [30, 18, 56, 64]. Though the concrete steps of these approaches differ and involve different physical qubits surrounding the lattice, all these approaches share the same elementary operation patterns, i.e., single-qubit operations on individual qubits and ESM. Since it is not clear which approach presents better performance, we will not consider how to support the detailed steps for rotating a planar surface code in this chapter and instead only ensure the elementary operation patterns can be supported by our envisioned microarchitecture.

LOGICAL S AND T GATES

The logical S and T gates on the rotated planar surface codes can be realized by performing a series of other logical operations, including the previously mentioned operations (measurement, Pauli, CNOT, and H) and another operation called magic state preparation. Magic states cannot be fault-tolerantly prepared using the transversal approach for logical state $|0\rangle$ or $|+\rangle$, they are prepared by two procedures: (i) *state injection* and (ii) *magic state distillation*. We refer readers to [15] for details and only summarize the elementary operation patterns required by magic state preparation as following:

- State injection is realized by applying physical operations on the data qubits of the lattice followed by several rounds of ESM.
- State distillation is implemented by logical circuits which only consist of logical preparation, measurement, Pauli, CNOT, and H.

SUMMARY

As a conclusion, the elementary operation patterns of all logical operations in these three categories can be classified into 1) physical operations on individual physical qubits, and 2) several rounds of ESM on lattices of which the size and base ancilla type can be adjusted at runtime. Since several rounds of ESM are required after most operations, a round of ESM is a frequently-used operation pattern, which implies ample space for optimization in the architectural and microarchitectural design. In the following sections, we will discuss how to efficiently support the execution of ESM and fault-tolerant operations in the control microarchitecture.

5.3. QUANTUM CONTROL MICROARCHITECTURE

To bridge the gap between quantum software and hardware, our previous work proposed the quantum control microarchitecture, QuMA [106]. QuMA can take as input instructions belonging to the quantum microinstruction set QuMIS, and generate the required signals to control superconducting qubits. To address the problem that QuMIS is too low-level and tightly bound to the hardware implementation, we upgraded QuMA into QuMA_v2 to support the executable QISA, eQASM [107]. eQASM, supported by QuMA_v2, features two kinds of runtime feedback, a Very-Long-Instruction-Word (VLIW) architecture, Single-Operation-Multiple-Qubit (SOMQ) execution, and flexible quantum operation definition. We briefly introduce the working principles of QuMA_v2 (Figure 5.1 excluding the dashed blocks) in this section and refer readers to [107] for detailed explanation.

QuMA_v2 is a heterogeneous architecture consisting of a conventional processor as the host, such as an Intel Xeon processor, and a quantum processor as the accelerator. The **host CPU** executes the classical part of the quantum program, and loads the quantum kernel(s) into the quantum processor for execution via the shared instruction memory and data memory. Besides quantum instructions, the quantum kernel can also contain *auxiliary classical instructions*. Auxiliary classical instructions are in charge of classical register update and program flow control including feedback based on qubit measurement results, which are processed by the **classical pipeline**. Quantum instructions, which are at the physical level, are sent to the **quantum instruction decoder** and following modules (together called the *quantum pipeline*) for further process.

The quantum pipeline can be divided into several stages to execute quantum instructions. The first stage, framed by the virtual address domain, decodes quantum instructions, resolves operation address and outputs as result quantum micro-operations for corresponding qubits. As a VLIW architecture, an eQASM instruction can contain multiple quantum operations, with each processed by a VLIW lane (**VLIW pipelane**). The **microcode unit** translates each quantum operation into one or multiple quantum micro-operations according to the control store content. The control store can be configured by the programmer, which enables flexible quantum operation definition at compile time. eQASM adopts SOMQ, an indirect address mechanism where every quantum operation takes a target register as the parameter, which stores a list of qubits or qubit pairs serving as the target of the quantum operation. The **quantum microinstruction buffer** resolves the target address, and assigns every quantum micro-operation to the corresponding target qubit(s). Because the timing of quantum operations is fully described by the instructions, every micro-operation is associated with a particular timing point generated from waiting instructions, which is handled by the **timestamp manager**. The **operation combination unit** merges the micro-operations from different VLIM pipelanes, and output them to corresponding qubits.

The second stage, framed by the physical address domain, mainly performs three tasks. First, operating a qubit may require the collaboration of multiple analog devices in the analog-digital interface, and a single analog device may also control multiple qubits. The **device event distributor** reorganizes micro-operations into *device operations* to trigger the corresponding devices. Second, device operations with the timing information are buffered at the queues of the **timing control unit** awaiting execution. The tim-

ing controller then triggers every device operation at its expected timing point. Third, the **fast conditional execution unit** allows executing or canceling single-qubit micro-operations based on a flag derived from the last measurement results of the same qubit using predefined combinatorial logic.

The **analog-digital interface** (ADI) is the boundary between the digital signals and analog signals. Once there comes micro-operations from the fast conditional execution unit, the **codeword triggered pulse generation module**, consisting of one or multiple analog devices, generates analog pulses to apply the corresponding quantum operations on the qubits. Regarding measurement, the **measurement discrimination unit** infers the measurement results of qubits based on the reflected analog waveform from the qubits, which results are used by the fast conditional execution unit and the classical pipeline for feedback.

Limited by the quantum operation issue rate, QuMA_v2 can only control tens of qubits. As mentioned previously, the amount of required qubits and operations can be significantly increased when implementing FTQC, which is a big challenge for the microarchitecture. To support efficient execution of massive quantum instructions required by FTQC, we envision a fault-tolerant control microarchitecture, FT-QuMA, based on QuMA_v2 , which is described in the next section.

5.4. FAULT-TOLERANT CONTROL MICROARCHITECTURE

Figure 5.1 gives an overview of the proposed fault-tolerant control microarchitecture FT-QuMA. It is upgraded from QuMA_v2 with the dashed blocks highlighting the modules and mechanisms introduced to support large-scale FTQC. The QECC chosen is the rotated planar surface code with logical operations implemented with lattice surgery.

As we have seen, each logical operation can be fault-tolerantly implemented by some physical operations and several rounds (or no rounds) of ESM. The fault-tolerant translation from logical operations to physical operations and ESM is explained in detail in [115], which can be done by the compiler. Then, instructions accepted by FT-QuMA are quantum operations at the physical level and ESMs on corresponding lattices which can be processed by the microarchitecture as explained in Section 5.4.2.

To make the compiler independent of actual physical qubit address, which enables a clear compilation model for large-scale quantum computing, FT-QuMA adopts *virtual memory* as used by classical computers. It supports fault-tolerant quantum computing by providing essential features for flexible logical operation description, efficient ESM circuit generation, quantum error decoding (QED), and Pauli frame, etc. Note, the user can turn off all modules related to QEC, and FT-QuMA can work properly as well, but at the physical level without support for fault-tolerant quantum computing. Modules shared by QuMA_v2 and FT-QuMA work in the same way unless otherwise specified.

5.4.1. QUBIT ADDRESSING

VIRTUAL ADDRESS AND PHYSICAL ADDRESS

To enable a simple compilation model for large-scale fault-tolerant quantum computing, two kinds of addresses are used: the virtual address and the physical address. It is assumed that the quantum compiler works with qubits on a virtual 2D array with NN

interaction among qubits. The virtual address has the format $q_v = (i, x, y)$, where i is the index address of the qubit and (x, y) is the Cartesian coordinates of the qubit on the virtual array. The index address is unique for every qubit and is used by quantum instructions and the microarchitecture to identify the target qubit(s) of each quantum operation. It can be determined just by counting qubits from bottom to top and from left to right in the virtual 2D array as shown in Figure 5.2. It can also be calculated from the Cartesian coordinates and vice versa, or a look-up table can be used to store the mapping between each other. The Cartesian coordinates are included as part of the address because they provide a straightforward method to determine what qubits are part of a logical qubit, the type of each qubit (data qubit, X ancilla, or Z ancilla), and the actual operations required by ESM. For instance, assuming a distance-d logical qubit is implemented by a square lattice, and the data qubit on the bottom left corner of this logical qubit has a virtual Cartesian coordinates of (x_0, y_0). Then all data qubits of this logical qubit can be found at (x_0+2i, y_0+2j), where $i, j \in \{0, 1, \cdots, d\}$. Knowing what kind of logical qubit is (see Figure 5.3), the locations of ancilla qubits and the corresponding types can also be determined in a similar way. In addition, the coordinates of neighbouring data qubits of each ancilla qubit as well as the physical operations required to perform ESM can be calculated, according to the circuits as shown in Figure 2.3b.

As shown in Figure 5.2, the virtual qubit array (red frame) is mapped to a lattice of the same size on the physical qubit array when initializing the quantum program on an actual quantum platform before execution. Every qubit also gets a physical qubit address $q_p = (\hat{i}, \hat{x}, \hat{y})$ at this step. Assuming the mapping process keeps the orientation of the axes of the qubit array, then the mapping can be determined by, e.g., recording the physical coordinates $q_p = (\hat{x_0}, \hat{y_0})$ of the virtual qubit (0, 0, 0). The Cartesian coordinates between the virtual address and the physical address can be translated using the relationship:

$$(\hat{x}, \hat{y}) = (x, y) + (\hat{x_0}, \hat{y_0}) \tag{5.1}$$

The physical address of qubits may vary when executing the program on different platforms. By using the virtual address, the compilation process can be independent of the actual physical qubit address, which contributes to a cleaner compilation model.

MICROARCHITECTURAL SUPPORT

The quantum instructions input to FT-QuMA use virtual addresses. As shown in Figure 5.1, modules in the virtual address domain works in the same way as explained in Section 5.3. They are relatively technology-independent and work with the virtual address. The virtual-physical address translation module translates the virtual address into the physical one according to Euqation (5.1). In the physical address domain, the modules are mostly technology-dependent and work with the physical address. In this way, the virtual address domain is clearly separated from the physical address domain. For instance, the measurement discrimination unit returns measurement results associated with physical addresses, which will be translated by the virtual-physical address translation module into virtual addresses and later sent to the classical pipeline for further process.

Note, the virtual address and physical address are only used by the physical qubits, and each logical qubit uses only one but unique logical identifier throughout the microarchitecture, which will be explained in the next subsection.

5.4.2. FAULT-TOLERANT LOGICAL OPERATIONS

As explained in Section 5.2.2, the elementary operation patterns required by the logical operations include physical operations on individual physical qubits, and several rounds of ESM on lattices. The keys to achieve fault-tolerance are to 1) measure the active stabilizers (ESM), and 2) decode and correct the errors at runtime. Lattice surgery needs to turn on (start to measure) and off (cease to measure) some stabilizers at different QEC cycles, which should be supported by the microarchitecture. Besides, the microarchitecture should be able to decode quantum errors at real-time as well.

Targeting these goals, FT-QuMA maintains a qubit symbol table (Q symbol table) that keeps track of the status of all logical qubits and it is used to determine active stabilizers. In addition, as we will explain in the following sections, FT-QuMA includes a QEC cycle generator, a quantum error decoder module and a logical measurement unit. It also includes a Pauli frame unit for keeping track of quantum errors on data qubits at runtime, which is well explained in [109] and will not be further explained here for brevity.

QUBIT SYMBOL TABLE

The microarchitecture should be able to know what logical qubits are being used (active), where they are, and the type of each physical qubit (data, X-ancilla, or Z-ancilla). The qubit symbol table (see Table 5.1) tracks which logical qubits are *active*, with '1' and '0' representing active and inactive, respectively. As shown in Figure 5.2, there are three active logical qubits, L_1, L_2, and L_3, where the subscription is the logical identifier (logical QID).

To generate correct instructions for ESM, the following configuration of logical qubits should be updated at runtime:

- Since a logical qubit can vary its size during lattice surgery, the *size* ($d_1 \times d_2$) *(size* column in Table 5.1) of each logical qubit should be recorded, where d_1 and d_2 are the number of **data qubits** of this logical qubit along x- and y-axis, respectively. As shown in Figure 5.2, logical qubit L_1 has a size of 3×3 (orange dots) while logical qubit L_3 has a size of 3×6.

- The *location* of each logical qubit *(location* column in Table 5.1) is essential to determine what physical qubits are used to implement such a logical qubit. The location of logical qubit L_i can be determined by recording the virtual coordinates (X_{L_i}, Y_{L_i}) of the **data qubit** at the bottom left corner of this logical qubit. For example, the location of logical qubit L_1 is $(2, 2)$.

- A logical qubit with the same data qubits can use different ancilla qubits for different purposes in lattice surgery. Take the distance-3 surface code as an example and as shown in Figure 5.3, there are in total four different flavors for the same logical qubit, which can be distinguished using two flags: the *base ancilla type* and the *chirality*. This information is required to determine what physical qubits are used as ancilla and what types they are.

Lattice surgery requires performing joint measurements over nearby logical qubits to realize a logical CNOT gate. As explained in Section 5.2.2, the final result of the joint

measurement is the product of measurement results of the new X- or Z-stabilizers introduced by the merging operation. To enable the logical joint measurement and quantum error decoding, which two logical *children* qubits (*children* column in Table 5.1) are merged into which larger *parent* logical qubit (*parent* column in Table 5.1) should be recorded at runtime as well. For instance, logical children qubits 1 and 2 in Table 5.1 are merged into their parent qubit 4.

An example of the qubit symbol table for the logical qubits in Figure 5.2 is shown in Table 5.1. When necessary, another table could be provided to inversely map the physical or virtual qubit addresses to logical identifiers.

Table 5.1: An example of the qubit symbol table, with the content recording the status of logical qubits in Figure 5.2.

Logical QID	Active	Location	Size	Base Ancilla Type	Chirality	Parent	Children
1	1	(2, 2)	(3, 3)	X	Left	4	-
2	1	(2, 8)	(3, 3)	X	Right	4	-
3	1	(10, 2)	(3, 6)	Z	Right	-	-
4	0	(2, 2)	(3, 6)	X	Right	-	[1, 2]

Based on the qubit symbol table, the hardware can deduct which physical qubits are used by each logical qubit, and the type of each physical qubit. Furthermore, all operations required by ESM can be determined according to the ESM circuits as shown in Figure 2.3b.

QEC CYCLE GENERATOR

As explained in Section 5.2, fault-tolerant logical operations are performed by performing some physical operations followed by several rounds of ESM. In addition, ESM is also required by QEC. As a result, most of the physical operations are used to perform ESM. Before discussing the QEC cycle generator, we will give a rough estimation of the number of physical operations required for ESM on all logical qubits.

In the following estimate, each logical qubit is encoded in a rotated planar surface code with a distance of d. Such a logical qubit consists of $d \times d$ data qubits and $d^2 - 1$ ancilla qubits for measuring X- or Z-stabilizers. As d increases, the weight-2 stabilizers only occupies around $2/d$ of all stabilizers, and most stabilizers are weight-4 that run the corresponding ESM circuit consisting of eight or six physical operations (Figure 2.3b). During lattice surgery, some stabilizers on the edge of logical qubits can be turned on or off, which may slight increase or decrease the number of physical operations required by ESM. For simplicity, it is assumed the total number of physical operations for ESM is not affected by lattice surgery. ESM should be performed with the active stabilizers of logical qubits repeatedly at every cycle, no matter a logical operation is being performed or not. Hence, instead of estimating the number of physical operations for ESM during the entire program, we only need to roughly estimate the physical operations required to perform one round of ESM:

$$N_{\text{op}}^{\text{C}} \approx N_p \times N_{\text{L}} \times d^2, \tag{5.2}$$

where N_{L} is the number of logical qubits and $N_p = 7$ is the average number of physical operations in each ESM circuit as shown in Figure 2.3b.

As can be seen, the number of physical operations and then instructions that need to be generated, explode as the d and/or N_L increase. Though the quantum compiler can generate the instructions for these physical operations, it cannot scale up because the quantum operation issue rate is a bottleneck for the QuMA microarchitecture [106]. In other words, the required instructions are too many to be fetched from the instruction memory for processing due to a limited instruction bandwidth. It highly requires the microarchitecture to generate physical operations for ESM to alleviate the quantum operation issue rate problem.

FT-QuMA introduces a special instruction, Gen_ESM L_i, which triggers the hardware to generate all physical operations to perform one round of ESM over the logical qubit L_i. This method is feasible based on the following observation. Given the location and size of a logical qubit in the qubit symbol table, the data qubits of this logical qubit can be determined. Based on the base ancilla type and chirality, the ancilla qubits and their type (hence the stabilizers) can be determined. Then, the microarchitecture can automatically generate a series of physical quantum operations with expected timing to implement the ESM circuit for each stabilizer according to Figure 2.3b. Note that, virtual (and then physical) index addresses are used in these quantum operations.

While translating the original quantum algorithm into the the fault-tolerant implementation, the compiler generates a Gen_ESM L_i instruction for every round of ESM on every active logical qubit in the binary. Once Gen_ESM L_i instruction is fetched during execution, this instructions triggers the QEC cycle generator after instruction decoding. The QEC cycle generator reads the location, size, base ancilla type, and the chirality of the target logical qubit L_i, based on which the physical operations with precise timing for ESM can be generated and sent to the timing control unit awaiting execution. By using the proposed method, the required number of instructions (N_{ins}^{M}) used for one round of ESM is substantially reduced to

$$N_{ins}^{M} \approx N_L. \tag{5.3}$$

The comparison between these two methods is shown in Figure 5.4. When $d = 30$, N_{ins}^{M} is four orders of magnitude less than N_{op}^{C}.

It is worth noting that, Tannu *et al.* [116] also proposed a similar idea, which adopts a microcode-based method to generate the instructions for ESM in the hardware. The microcode-based method can support not only planar surface codes but also defect-based surface codes.

5.4.3. QUANTUM ERROR DECODING

After a particular number of rounds of ESM have been applied on a logical qubit, the quantum error decoder is responsible for inferring the possible errors based on the syndromes, i.e, the stabilizer measurement results. Although the Pauli frame can keep track of the identified errors, the decoding process must be as fast as possible, because the qubit measurement results stored in the measurement result unit should contain the correct values after error correction to support runtime feedback. For example, in an implementation of the Shor's algorithm [117], some X gates are conditioned on the results of previous measurements to reduce the number of required qubits. Therefore, the error decoding process is required to be fast enough so that the computation will not

be delayed, leading to more accumulated errors or even a failure of the computation. Ideally, the quantum error decoder decodes the error syndromes generated by d rounds of ESM using less time than the duration of these ESM circuits. To achieve such high speed, the decoding unit can decode errors based on, e.g., Blossom algorithm [110] or neural networks [36, 112, 111].

In order to support quantum error decoding, FT-QuMA introduces another special instruction, Decode_ESM L_i type. It triggers the quantum error decoder to perform decoding over the type syndromes of the last rounds of ESM for the logical qubit L_i, where type can be X, Z, or XZ. The syndrome type should be specified in the decoding instruction because the decoding lattices for X- and Z-syndromes may be different for a lattice surgery-based operation (Section 5.2.2). Once the Decode_ESM L_i type instruction is fetched during execution, the QEC cycle generator reads the entry of the logical qubit L_i in the qubit symbol table, which information is finally sent to trigger the quantum error decoder at the expected timing point by the timing control unit. The quantum error decoder contains multiple instances of the decoding unit. Each decoding instance is triggered by one Decode_ESM L_i type instruction with the logical qubit information. Based on this information, the decoding instance can determine which physical qubits are used as X- or Z-ancilla qubits, and detect the errors that may happen in that logical qubit. Note, a distributed implementation of the quantum error decoder can be easily parallelized and has better scalability.

5.4.4. MEASUREMENT RESULT UNIT

There are two kinds of logical measurements used by planar surface codes based on lattice surgery: the measurement of a single logical qubit and the joint measurement of two logical qubits. (1) The measurement of a single logical qubit is realized by measuring all the data qubits and later classically checking the parity of these data qubits to perform error correction. After removing the detected errors, the logical measurement result is calculated by multiplying the measurement outcomes of all data qubits of the logical qubit. (2) The joint measurement of two logical qubits is realized by a merge operation followed by a split operation using lattice surgery (Section 5.2.2). After error correction, the joint measurement outcome is calculated by multiplying the outcomes of some newly measured stabilizers.

To be able to perform both measurements, two measurement instructions are defined. The Single_Msmt_Result L_i instruction is used to trigger the measurement result unit to discriminate the result of measuring the logical qubit L_i. After decoding, address translation and awaiting execution, every Single_Msmt_Result L_i instruction fetched from the instruction memory finally triggers the measurement result unit at a precise timing to perform error correction, and calculates the logical measurement result for logical qubit L_i. Note that the entry of this logical qubit in the qubit symbol table is also sent to the measurement result unit.

To support the joint measurement of two logical qubits, the instruction Joint_Msmt_ Result L_i, L_j is introduced. After performing error correction for a joint measurement, the Joint_Msmt_ Result L_i, L_j instruction triggers the measurement result unit to multiply the outcomes of newly measured stabilizers in the lattice merged from L_i and L_j.

The measurement result can be read by the classical pipeline for feedback, e.g., di-

recting the following program flow. The result can also be sent to the host CPU via the data memory. Note that the logical measurement unit also stores the latest measurement result of each physical qubit to enable feedback at the physical level.

5.5. LATTICE SURGERY INSTANTIATION

In this section, we illustrate how the proposed control microarchitecture can efficiently support surface-code-based FTQC by lattice surgery. The joint measurement M_{ZZ} between qubits 'A' and 'C' in Figure 4.1 using the control microarchitecture is performed as following:

1. First, the status of these two logical qubits before the joint measurement is shown in Table 5.2. Then d Gen_ESM L_i instructions are issued. Each instruction triggers the generation of physical operations for one round of ESM on active logical qubit L_i ($i \in \{1,2\}$). Afterwards, the instruction Decode_ESM L_i XZ will trigger the quantum error decoder to decode errors for logical qubit L_i ($i \in \{1,2\}$).

2. In order to perform the merging operation in Figure 4.1b, the status of logical qubits will be updated into the ones in Table 5.3. Similarly, d Gen_ESM L_4 instructions will be issued to perform d rounds of ESM on the active qubit in Table 5.3, i.e., L_4. Note that the decoding for merging and splitting operations are different from the general QEC procedure. To perform fault-tolerant decoding for the merging operation, instructions Decode_ESM L_4 X, Decode_ESM L_1 Z, and Decode_ESML_2 Z will be issued. Moreover, the instruction Joint_Msmt_Result L_1, L_2 will trigger the measurement result unit to calculate the result of joint measurement M_{ZZ}.

3. To perform the splitting operation in Figure 4.1c, the qubit symbol table should be reverted to the status as shown in Table 5.2. Then d Gen_ESM L_i instructions should be issued for each logical qubit ($i \in \{1,2\}$) to trigger d rounds of ESM. The decoding process for the splitting operation is the same as the merging operation. Hence, the same three instructions will be issued, i.e., instructions Decode_ESM-L_4 X, Decode_ESM L_1 Z, and Decode_ESM L_2 Z. Note, even though the logical qubit L_4 is inactive, the decoding for Z-syndromes should be still performed over the lattice corresponding to L_4.

After steps (1) to (3) have been done, the joint measurement over logical qubit L_1 and L_2 is completed.

Table 5.2: The initial status of logical qubits 'A' and 'C' before performing a merging operation in Figure 4.1a.

Logical QID	Active	Location	Size	Base Ancilla Type	Chirality	Parent	Children
1 ('C')	1	(2, 2)	(3, 3)	X	Left	4	-
2 ('A')	1	(2, 8)	(3, 3)	X	Right	4	-
4 ('AC')	0	(2, 2)	(3, 6)	X	Right	-	[1,2]

Table 5.3: The status of logical qubits 'A' and 'C' after performing a merge operation in Figure 4.1b.

Logical QID	Active	Location	Size	Base Ancilla Type	Chirality	Parent	Children
1 ('C')	0	(2, 2)	(3, 3)	*X*	Left	4	-
2 ('A')	0	(2, 8)	(3, 3)	*X*	Right	4	-
4 ('AC')	1	(2, 2)	(3, 6)	*X*	Right	-	[1,2]

5.6. DISCUSSION AND CONCLUSION

A quantum instruction set architecture with a quantum control microarchitecture is required to bridge the gap between quantum software and quantum hardware. Previous research in this field mainly focused on NISQ devices with relatively low number of qubits and no quantum error correction is used. However, as the number of qubits increases, quantum error correction and fault-tolerant mechanisms will be included to achieve reliable quantum computation.

In This chapter, we present a microarchitecture, FT-QuMA, to support the execution of fault-tolerant logical operations for a specific QEC code (the rotated planar surface code) with logical operations based on lattice surgery. We adopt the concept virtual memory from classical computing to quantum computing to provide a clean compilation model, which is independent of the actual physical addresses of qubits that can vary from device to device. With the support of ESM circuit generation and quantum error decoding at runtime, the codesize of the fault-tolerant implementation of the quantum program can be significantly reduced.

Our proposal is an early-stage attempt to define the architectural blocks and instructions required to perform fault-tolerant quantum computation. Hence, there can be better designs of the entire architecture and these architectural blocks, which can be used after a comprehensive evaluation. In addition, FT-QuMA has been specifically designed for rotated planar surface codes and lattice-surgery-based operations, and will have some limitations if other QEC schemes are used as we will discuss in this section.

One of the main questions when designing the interface between software and hardware is how to divide the task between them. There is a trade-off between what tasks to allocate on the compiler and on the microarchitecture. In this chapter, we propose that the compiler translates fault-tolerant logical operations to physical operations and ESM instructions. The main advantage of this approach is its flexibility. When using a different way to implement fault-tolerant logical operations or even a different QEC code, modification is only required in the software to change the routine which decomposes logical operations into corresponding lower-level operations. As aforementioned, other approaches are possible. For instance, the compiler could directly output logical instructions, leaving the task of generating the corresponding physical operations as well as the ESM operations to the microarchitecture. This method further abstracts away the low-level details of the underlying qubit array, which leaves a simpler interface to the software. However, it poses a big challenge in the microarchitecture design to translate logical operations to physical operations, which is cumbersome to change to suit other logical operation implementation scheme. Hence, it is less flexible than the one proposed in This chapter.

In our design, the compiler also generates ESM instructions for every active logical

qubit as well as *decoding* instructions. Both of them might be also automatically generated by the microarchitecture, but it requires further design to ensure that instructions on different logical qubits are properly synchronized.

As mentioned, the proposed FT-QuMA has been designed for the rotated planar surface code and currently it can only support logical operations on regular rectangle layouts. Note that, some logical operations may change the lattices from regular rectangles to irregular shapes. In addition, other encoding schemes such as the twisted surface code and the defect surface code use a different implementation of logical operations. They may also operate on irregular lattice layouts, which will require to add some extra information in the qubit symbol table. What modifications should be made on the control microarchitecture for supporting these irregular shapes and other mechanisms when different QEC schemes are considered needs further investigation. Future work will also involve the development of the hardware blocks in FT-QuMA that support the virtual memory and fault-tolerant quantum computing, and verify FT-QuMA on our developing full-stack simulator, called quantum virtual machine, which can simulate not only the qubit state evolution but also the execution of quantum instructions on the microarchitecture.

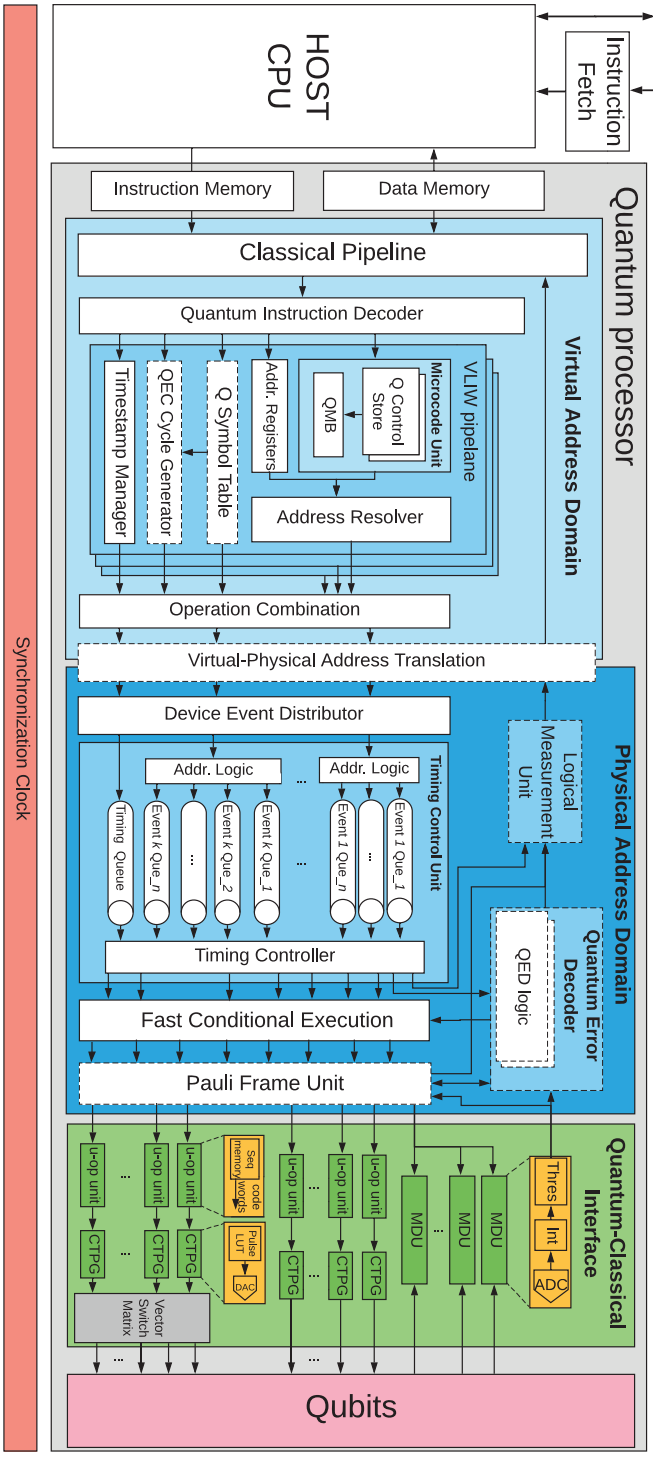


Figure 5.1: Overview of the Envisioned Fault-Tolerant Quantum Control Microarchitecture, FT-QuMA, which is upgraded from QuMA_v2, with the dashed blocks highlighting the changes. Black thin lines indicate digital signals and gray thick lines analog signals.

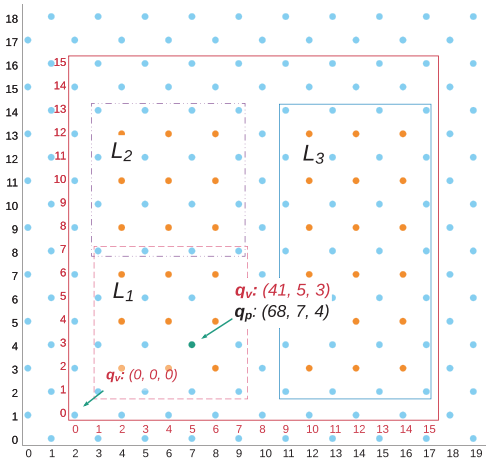


Figure 5.2: Virtual qubits mapped on physical qubits. Red is for the virtual address space and black for the physical address space.

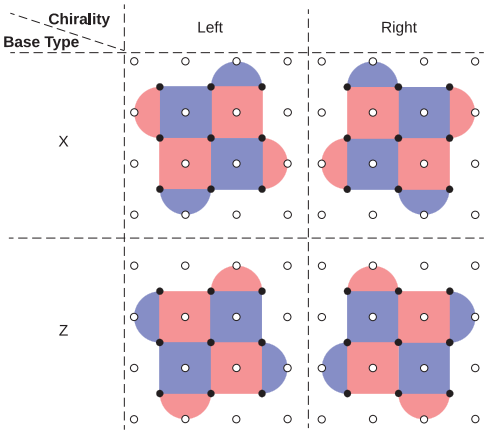


Figure 5.3: Four flavors of a distance-3 surface code logical qubit using the same data qubits. It is assumed that redundant ancilla qubits are not used. The chirality of the logical qubit is left (right) when the physical qubit at (X_L+1, Y_L-1) ((X_L+3, Y_L-1)) is used as an ancilla qubit, where (X_L, Y_L) is the location of the logical qubit. The base ancilla type is X (Z) when the ancilla qubit at (X_L+1, Y_L-1) or (X_L+3, Y_L-1) is an X (Z) ancilla.

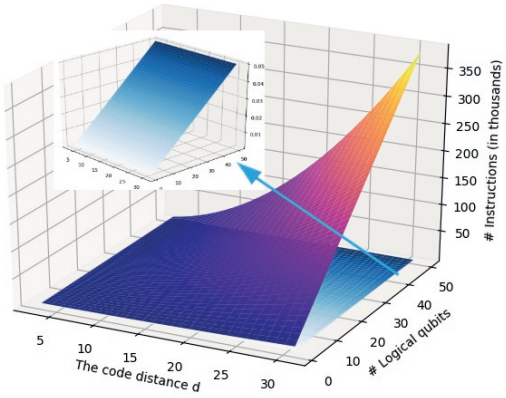


Figure 5.4: The number of instructions for performing one round of ESM on all active logical qubits through compiler generation (top) and hardware generation (bottom).

6

Fault-tolerant Quantum Error Correction on Near-term Quantum Processors using Flag and Bridge Qubits

Fault-tolerant (FT) computation by using quantum error correction (QEC) is essential for realizing large-scale quantum algorithms. Devices are expected to have enough qubits to demonstrate aspects of fault tolerance in the near future. However, these near-term quantum processors will only contain a small amount of noisy qubits and allow limited qubit connectivity. Fault-tolerant schemes that not only have low qubit overhead but also comply with geometrical interaction constraints are therefore necessary. In this chapter, we combine flag fault tolerance with qubit routing techniques to enable an efficient flag-bridge *approach to implement FT QEC on near-term devices. We further show an example of performing the Steane code error correction on two current superconducting processors and numerically analyze their performance with circuit level noise. The simulation results show that the QEC circuits that measure more stabilizers in parallel have lower logical error rates. We also observe that the Steane code can outperform the distance-3 surface code using flag-bridge error correction. In addition, we foresee potential applications of the flag-bridge approach such as FT computation using lattice surgery and code deformation techniques.*

This chapter is based on L. Lao, C. G. Almudever, *Fault-tolerant Quantum Error Correction on Near-term Processors using Flag and Bridge Qubits*, arXiv:1909.07628 (2019).

6.1. INTRODUCTION

Near-term quantum processors will consist of fifty to a few hundred noisy qubits and allow a limited number of faulty gates. They are also known as Noisy-Intermediate-Scale Quantum (NISQ) [21] processors. For instance, Google, IBM, and Intel have respectively announced 72-qubit [118], 50-qubit [119], and 49-qubit [120] superconducting processors which have coherence times of ~ 100 microseconds and two-qubit gate error rates near 0.1% [16]. Many efforts have been focusing on designing special quantum applications [121, 122] and developing compilation techniques [106, 123] such that one can solve practical problems and even demonstrate quantum supremacy on NISQ processors only using noisy bare qubits.

However, fault tolerance will be necessary to reliably implement large-scale quantum algorithms. This can be achieved through the use of active quantum error correction (QEC). The idea of QEC is to encode one logical qubit into many physical qubits and repeatedly perform syndrome extraction to detect and correct errors. Both the encoding and error detection procedure should be fault-tolerant (FT). Furthermore, operations on these logical qubits need to be performed fault-tolerantly. Although the high qubit overhead of QEC makes it difficult to realize scalable FT computation in the near future, we can begin to learn how fault tolerance works in practice. The first step is to demonstrate fault-tolerant quantum error correction, that is, FT quantum memory.

General fault-tolerant quantum error correction protocols such as those from Shor [124], Steane [125], and Knill [126] can be applied to various stabilizer codes. However, these error correction schemes all require many ancilla qubits, which are scarce resources in near-term quantum processors. In order to perform FT QEC with low qubit overhead, a new error correction protocol has been proposed [114, 127, 128, 129]. It replaces a non-FT syndrome extraction circuit by a circuit which can detect correlated (or hook) errors by adding only one or a few extra ancilla qubits, called **flag** qubits.

This flag QEC scheme provides an efficient way to demonstrate fault tolerance in small experiments. However, many orthodox flag circuits couple one qubit to many others, requiring high-degree qubit connectivity. It is difficult or even impossible to directly map available flag circuits onto near-term quantum processors which have geometrical interaction constraints such as the nearest-neighbor connectivity in superconducting processors [14, 39, 130]. One may need to apply extra operations such as SWAP gates to move qubits to be adjacent, increasing the circuit size in terms of depth and total gate number, or even circuit width. More importantly, the resulting circuit may not be fault-tolerant, or produce higher error rates when used.

In this chapter, we extend the set of available flag circuits to a variety of equivalent circuits that can perform the same stabilizer measurement fault-tolerantly. In these circuits, the flag qubits are also used as **bridges** to cope with the connectivity constraints, called **flag-bridge** qubits. Using these circuits, one can fault-tolerantly map a QEC code to a given processor with low overhead by choosing appropriate flag-bridge circuits. We also develop a simulation framework to automate the procedure of fault tolerance checking, decoder design (including a look-up-table decoder and a neural-network decoder) for given flag-bridge circuits of some low-distance QEC codes. This automation is desirable for demonstrating fault-tolerant quantum error correction in small experiments. Moreover, we present mapping examples of the Steane code on two different qubit pro-

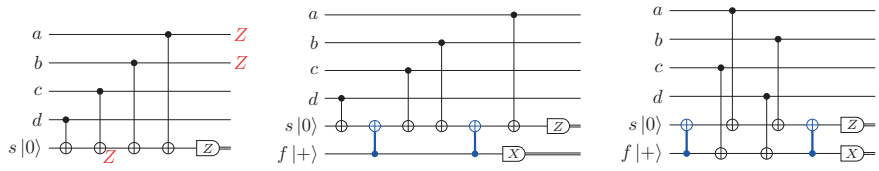
cessor topologies and analyze their fault tolerance numerically. In addition, we show the proposed flag-bridge approach can be applied to FT computation implemented by lattice surgery and code deformation techniques.

The rest of this chapter is organized as follows. We first review the basics of flag-based quantum error correction in Section 6.2. Then we introduce the proposed flag-bridge approach in Section 6.3. Afterwards, the mapping of the Steane code onto two qubit processor topologies and corresponding numerical results are shown in Section 6.4. Moreover, we provide the potential applications of flag-bridge circuits in Section 6.5. Finally, Section 6.6 concludes the chapter.

6.2. FLAG-BASED QUANTUM ERROR CORRECTION

In this section, we briefly introduce the flag-based error syndrome extraction for stabilizer codes. For more details, we refer the readers to [114, 127, 128, 129, 131].

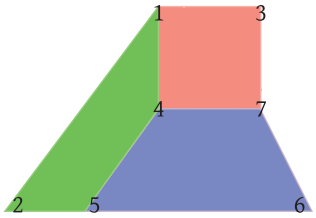
Figure 6.1 shows the circuits for measuring a weight-4 Z-stabilizer (or check), similar circuits can be derived for measuring other Pauli operators. In all the circuits presented in this chapter, a CNOT gate between a data qubit and an ancilla qubit is called an s-CNOT (in black) and a CNOT gate between two ancilla qubits is called an f-CNOT (in blue). Generally, the syndrome for this Z-check can be extracted using the circuit with only one ancilla qubit (Figure 6.1a). However, this circuit is not fault-tolerant because one single fault could cause 2 or more data errors. These correlated errors may lead to failures of some QEC codes. The surface code is an exception which can correct these hook errors if the two-qubit gates are performed in a specific order [15]. In order to perform fault-tolerant quantum error correction, one can use the flag circuits in Figures 6.1b and 6.1c that only add one extra ancilla qubit. When there is no fault, each of these flag circuits behaves the same as the non-FT one. When there is a fault that can lead to hook errors, it will yield a non-trivial measurement outcome of the flag qubit such that the hook errors are detected. For instance, if the same fault in Figure 6.1a happens in the circuit of Figure 6.1b, then the measurement of qubit f will be '1' (raising a flag).



(a) (b) (c)

Figure 6.1: The syndrome extraction circuits for the $Z_{a,b,c,d}$ operator, where s is the syndrome qubit and f is the flag qubit. (a) The circuit only using one syndrome ancilla may not be fault-tolerant. For example, one fault (Z_s) on the second CNOT gate could lead to correlated weight-2 errors on data qubits (Z_a, Z_b), which may not be correctable. (b) and (c) The flag-based circuits can detect these hook errors [114, 127, 128].

Flag-based quantum error correction can be applied to many codes such as the $[[5,1,3]]$ code, Hamming codes, surface codes, color codes, etc. For example, fault-tolerant QEC for the smallest color code, the Steane code in Figure 6.2, can be realized as follows: first measure each stabilizer generator one by one using flag circuits similar to those in Fig-



$$S_X^1 = X_{1,2,4,5} \quad S_Z^1 = Z_{1,2,4,5}$$
$$S_X^2 = X_{1,3,4,7} \quad S_Z^2 = Z_{1,3,4,7}$$
$$S_X^3 = X_{4,5,6,7} \quad S_Z^3 = Z_{4,5,6,7}$$
$$\overline{X} = X_{1,2,3,4,5,6,7} \quad \overline{Z} = Z_{1,2,3,4,5,6,7}$$

Figure 6.2: (Left) The qubit layout of the ⟦7, 1, 3⟧ Steane code. Data qubits are on the vertices and each plaquette represents two stabilizers: one weight-4 X-stabilizer and one weight-4 Z-stabilizer. (Right) all the six stabilizer generators and logical X and Z operators.

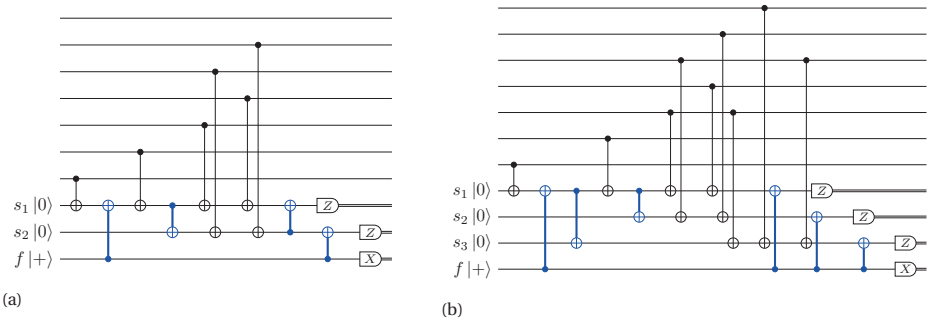


Figure 6.3: Flag circuits [127, 129] for (a) measuring two weight-4 Z-checks in parallel using three ancillas and (b) measuring three weight-4 Z-checks in parallel using four ancillas.

ure 6.1; if a flag raises or a syndrome appears, then stop this round[1] and sequentially measure all the stabilizers using the non-FT syndrome extraction circuit. Note that if connectivity is fixed, we cannot necessarily change the syndrome measurement circuit all of a sudden. One can use only two ancilla qubits to perform FT QEC for the Steane code at the cost of using more time steps. However, many quantum systems have very short coherence times [4, 3, 14]. Parallelizing stabilizer measurement will be beneficial to achieve lower logical error rates. Chao and Reichardt [127, 129] have proposed several circuits to perform two or three parity checks in parallel for the Steane code. The circuits they propose for measuring two and three Z-checks at the same time using only one flag qubit are shown in Figure 6.3. As shown, more ancilla qubits are required to achieve this parallelism compared to the sequential stabilizer measurement circuits. This implies there is a trade-off between the number of qubits required and the number of stabilizers that can be measured simultaneously.

Flag-based syndrome extraction is promising for demonstrating quantum error correction and fault tolerance in small quantum experiments because of its low qubit overhead. However, current or near-term quantum processors have many hardware limitations. One of the main constraints is the degree of qubit connectivity, that is, one qubit

[1] A full round of error syndrome extraction is defined as measuring all the stabilizer generators of the code for one time.

can only interact with a limited number of other qubits. It is challenging to map existing flag circuits onto connectivity-constrained processors meanwhile maintaining the fault tolerance with low costs. For instance, the ancilla qubit s of the flag circuit in Figure 6.1b needs to interact with five qubits, which cannot be supported in a grid topology where each qubit only has at most four neighbors such as the one in [39]. In this chapter, we propose a flag-bridge approach to solve this mapping problem, which will be explained in the next section.

6.3. FLAG-BRIDGE QUANTUM ERROR CORRECTION

In this section, we illustrate the proposed flag-bridge approach which allows fault-tolerant quantum error correction with low qubit overhead on connectivity-limited quantum processors.

6.3.1. FLAG-BRIDGE SYNDROME EXTRACTION CIRCUITS

We provide a microscopic explanation of how a flag-based circuit can perform a specific stabilizer measurement using the stabilizer formalism [25]. We will use the circuit in Figure 6.1c as an example. A flag syndrome extraction circuit can be understood as a circuit that replaces the bare ancilla qubit by an 'encoded' ancilla up to gate commutation. As shown in Figure 6.1c, the first blue CNOT gate entangles ancilla qubit s and qubit f (the encoding procedure), encoding a logical ancilla in a $[\![2, 1, 1]\!]$ error detection code whose stabilizer is

$$\langle X_s \otimes X_f \rangle$$

and logical operators are

$$\left\langle \overline{X} = X_s, \overline{Z} = Z_s \otimes Z_f \right\rangle.$$

This logical qubit is fixed in the $\overline{Z}$ basis. Then one can perform stabilizer measurement using this logical ancilla. Assume the four data qubits (a, b, c, d) are initially stabilized by $(-1)^y Z_{a,b,c,d}$ $(Z_a \otimes Z_b \otimes Z_c \otimes Z_d)$, the four subsequent s-CNOT gates between data qubits and ancilla qubits will keep the stabilizers of all the qubits invariant, which are,

$$\langle X_s \otimes X_f, (-1)^y Z_{4,5,6,7} \rangle,$$

but it will gradually transform the logical operators into

$$\left\langle \overline{X} = X_s, \overline{Z} = Z_s \otimes Z_f \otimes (-1)^y Z_{4,5,6,7} \right\rangle.$$

More generally, since X_f and X_s have the same effect on the encoded ancilla state, one can perform each s-CNOT gate between the particular data qubit with any ancilla qubit. Specifically, in the encoded ancilla area, k_s and k_f s-CNOT gates can be applied on ancillas s and f respectively, where k_s and k_f are integers and $k_s + k_f = 4$. For example, the circuit shown in Figure 6.4a also performs a weight-4 $Z-$stabilizer measurement equivalent to this circuit (Figure 6.1c), where $k_s = 3$ and $k_f = 1$.

Afterwards, the last f-CNOT (the decoding circuit) disentangles these two ancillas, leading to the final stabilizer to be

$$\langle (-1)^y Z_{4,5,6,7} \rangle$$

and the logical operators of these ancillas to be

$$\langle X_f, (-1)^y Z_s \rangle.$$

This means the readout y of measurement M_z on ancilla s indicates the measurement result of the stabilizer $Z_{a,b,c,d}$. Therefore, this circuit indeed measures a weight-4 Z-check. Besides, the measurement result of ancilla f implies the syndrome of the $[\![2, 1, 1]\!]$ code, that is, it can detect one single Z error that occurs on any ancilla and then raises a flag.

Once a flag circuit based on the above approach is generated, one can transform it into other equivalent ones that can perform the same stabilizer measurement by applying gate commutation, e.g., the circuit in Figure 6.1b. Note that the circuits generated by commuting gates may not be fault-tolerant. Moreover, one can use a larger 'encoded' ancilla to measure a weight-n Z-check (similar circuits can be applied to other Pauli operators). This logical ancilla is encoded by m physical qubits denoted by a set $Q = \{1, 2, \cdots, m\}$, where one is syndrome qubit ($Q_s = \{1\}$) and the other $m-1$ are flag qubits ($Q_f = \{2, \cdots, m\}$). The underlying error detection code $[\![m, 1, 1]\!]$ has stabilizers

$$\langle X_j \otimes X_k \rangle, j \in Q_s, k \in Q_f$$

and logical operators

$$\left\langle \overline{X} = X_j, \overline{Z} = \bigotimes_i Z_i \right\rangle, i, j \in Q.$$

Similar to the two-ancilla flag circuits, this weight-n check can be distributed to all m ancillas, k_i s-CNOT gates will be applied on ancilla i, where $\sum_{i=1}^{m} k_i = n$. For example, the circuit in Figure 6.4b measures one weight-4 Z-stabilizer using one syndrome qubit (s) and two flag qubits (f_1, f_2), each qubit only needs to interact with at most three others.

In addition, one can also measure p Z-checks in parallel by encoding p logical ancillas into m physical ancillas. The underlying $[\![m, p, 1]\!]$ code is stabilized by

$$\left\langle X_i \otimes \bigotimes_j X_j \right\rangle, i \in Q_f, j \in Q_s.$$

Its p logical operators are

$$\left\langle \overline{X_k} = X_i, \overline{Z_k} = Z_i \otimes Z_j \right\rangle, i, j \in Q, i < j.$$

Where, Q_s is the set of p syndrome qubits and Q_f is the set of $m-p$ flag qubits. After the encoding of ancilla qubits, one can simply assign all the s-CNOT gates for performing one check to a particular syndrome qubit. In this parallel syndrome extraction case, one can reduce the total number of s-CNOT gates by applying gate commutation when two or more checks are performed on the same data qubit(s). Figure 6.3a and Figure 6.3b show the flag circuits to measure two and three checks of the Steane code in parallel by using ancillas encoded in a $[\![3, 2, 1]\!]$ code and in a $[\![4, 3, 1]\!]$ code, respectively, These circuits use fewer s-CNOT gates than required by commuting some CNOT gates out of the encoded area (Generally 4 s-CNOT gates are needed for each weight-4 check).

Moreover, one can achieve this gate reduction by distributing some s-CNOT gates to flag qubits, which can even help to reduce the circuit depth. Figure 6.5a shows the example circuit that measures two checks of the Steane code in parallel but uses fewer timesteps than Figure 6.3a. Note that the s-CNOT distribution for parallel syndrome measurement needs to be designed carefully since one flag qubit is used for flagging multiple checks. This distribution also depends on the decoding procedure of the $[\![m, p, 1]\!]$ code. Figure 6.5b shows the example circuit that measures three checks using fewer timesteps than Figure 6.3b.

By employing the ideas of encoding ancillas, distributing s-CNOT, and commuting gates, we can generate more equivalent syndrome extraction circuits that have different connectivity requirements. Note that not all the equivalent circuits generated using this approach are fault-tolerant. The fault tolerance can be checked based on the error correction protocol, which will be explained in the next section. For these FT circuits, ancillas are not only used as syndrome and flag qubits to detect errors, but also as bridges to allow the interaction between data qubits and the encoded ancilla block. Such a syndrome extraction circuit is called a flag-bridge circuit.

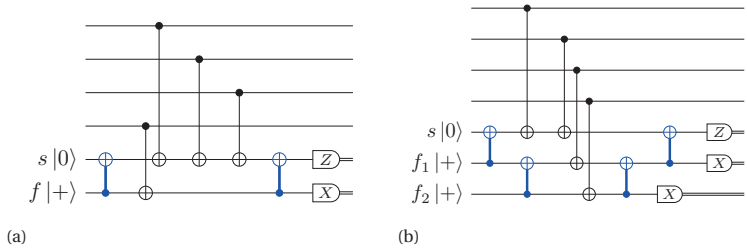


Figure 6.4: Flag-bridge circuits for measuring one weight-4 Z-check using (a) two ancillas and (b) three ancillas.

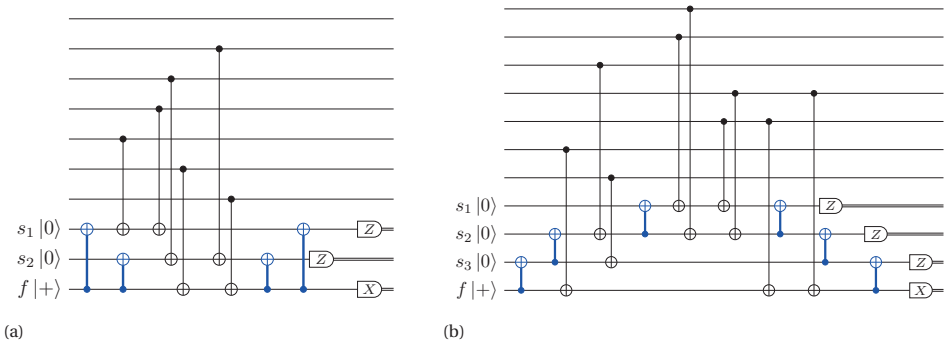


Figure 6.5: Flag-bridge circuits which measure (a) two and (b) three weight-4 Z-checks in parallel.

6.3.2. FAULT-TOLERANT PROTOCOL FOR FLAG-BRIDGE ERROR CORRECTION

FT QEC CONDITION

For distance-3 codes, a QEC circuit is fault-tolerant if it can either immediately correct all errors from a single fault or only leave a weight-1 error to the next cycle. A formal condition of FT flag-bridge quantum error correction for distance-3 codes, similar to the flag error correction in [128], can be defined as follows:

Consider a stabilizer code $\mathscr{S} = \langle g_1, g_2, \cdots, g_r \rangle$ and its QEC circuit $\mathscr{C}$ which is composed of the flag-bridge circuits for measuring the stabilizer generators, that is, $\mathscr{C} = \{c(g_1), c(g_2), \cdots, c(g_r)\}$, where $c(g_i)$ is the flag circuit of measuring stabilizer g_i. Note that the total number of flag-bridge circuits is smaller than r if several stabilizers are measured simultaneously in one flag-bridge circuit. For all generators g, all pairs of elements $E, E' \in \mathscr{E}(g)$ satisfy $sf(E) \neq sf(E')$ or $E \sim E'$, where $\mathscr{E}(g)$ is the set of all errors caused by one fault, $sf(E)$ is the syndrome and flag string caused by E. We define $E \sim E'$ to mean that there is an element g in $\mathscr{S}$ such that $E' \propto gE$, that is, these errors are stabilizer-equivalent.

Based on this criterion, we check the fault tolerance of each generated QEC circuit $\mathscr{C}$ through a brute-force simulation under circuit level noise, analogous to [114]. It is implemented by injecting each individual fault from a circuit-based error model on every single-qubit or two-qubit gate in a given QEC circuit and then collecting the final syndromes and flags. If there are two or more sets of errors which lead to the same syndrome-flag string but do not yield a stabilizer when multiplied, then this QEC circuit is not fault-tolerant.

FT QEC PROCEDURE

A full cycle of fault-tolerant error correction for distance-3 codes using flag-bridge circuits can be performed as follows:

1. For the first round of syndrome extraction, each circuit $c(g_i) \in \mathscr{C}$ is sequentially performed. If there are non-trivial flags f_i^1 or non-trivial syndromes s_i^1 of $c(g_i)$, then this round will be terminated and another full round for all circuits in $\mathscr{C}$ will be performed. All the syndromes $s^2 = \bigcup_i s_i^2$ and flags $f^2 = \bigcup_i f_i^2$ of the second round will be collected.

2. If f_i^1 is not empty, one can decode using f_i^1 and s^2 (and f^2). If f_i^1 is empty, but s_i^1 is not empty, one can decode using s^2 (and f^2). Otherwise, no corrections are needed.

In this FT QEC procedure, we use flag-bridge circuits for both rounds of syndrome extraction because of the connectivity constraint, which is different from the ones proposed in [127, 128, 129], where non-FT syndrome extraction circuits that use only one ancilla are executed for the second round.

ERROR DECODERS

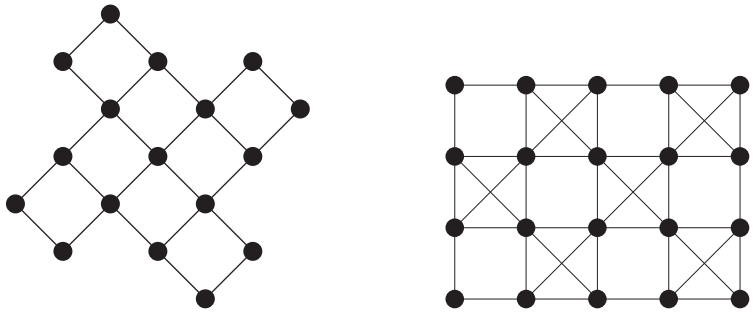
Normally, error correction of topological codes like surface codes have special structures for the measured syndromes so that one can use heuristic algorithms to find high-probability errors. These types of decoders such as the minimum weight perfect matching decoder [132] and the belief propagation decoder [133] can be applied to the same QEC code with different distances. However, the flag-bridge error correction circuits of a

QEC code for a specific quantum platform are ad hoc. Different circuits may be chosen based on the qubit topology, leading to different error-syndrome patterns and in turn requiring different decoding strategies. It is difficult to design heuristic decoding algorithms that can be applied to various syndrome extraction circuits. Since flag-bridge circuits are likely to be used for low-distance codes in small experiments, a simple decoding solution is to create a look-up table (LUT) for each QEC circuit. A LUT decoder can find the most likely Pauli errors from a single fault that leads to the observed syndromes and flags. LUT decoders can be easily derived from the brute-force checking procedure [114].

Another type of decoders are the neural-network (NN) decoders [36, 37, 134, 135]. They can provide high-speed decoding, be adaptable to different error models, and be more easily implemented on hardware. Moreover, a NN decoder can be developed by training the network using only input-output pairs without any knowledge of the QEC code, making it favorable for flag-bridge circuits. For example, the inputs of a NN decoder are the observed syndromes and its outputs can be the actual physical errors that have occurred. The implementation details of the LUT decoder and the NN decoder can be found in Section 6.7.

In this chapter, we design a simulation framework to automate the procedure of fault tolerance checking, LUT generation, and NN decoder training for given flag-bridge syndrome extraction circuits of the Steane code. This automation is desirable for demonstrating fault-tolerant quantum error correction in near-term processors which may have different geometrical interaction constraints.

6.4. STEANE CODE ERROR CORRECTION ONTO TWO QUBIT PROCESSOR TOPOLOGIES



(a) (b)

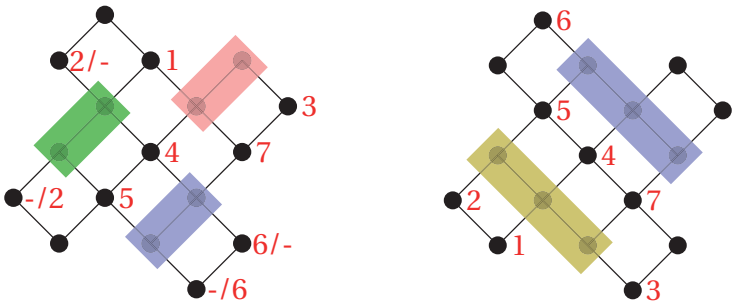
Figure 6.6: (a) The Surface-17 topology and (b) the IBM-20 topology, where each node represents a qubit, and each edge indicates the connectivity between two qubits.

In this section, we show how to map the Steane code error correction onto two different processors with limited connectivity using the proposed flag-bridge circuits, namely, the Surface-17 transmon processor (Surface-17) [39] and the IBM Q Tokyo processor (IBM-20) [14] (Figure 6.6). Furthermore, we numerically analyze each flag-bridge quan-

tum error correction procedure under circuit level noise. This error model inserts depolarizing errors after each operation in a flag-bridge circuit as follows: 1) each single-qubit gate is followed by a X, Y, or Z with probability $p/3$; 2) each two-qubit gate is followed by an element of $\{I,X,Y,Z\}^{\otimes 2}\backslash\{II\}$ with probability $p/15$; 3) the preparation or measurement in the Z basis is flipped with probability p. The elementary Clifford operations used in this simulation are preparation and measurement in the Z basis, H and CNOT gates. Other operations need to be further decomposed into these elementary operations. For example, each control-phase gate is replaced by two H gates and one CNOT gate.

6.4.1. MAPPING

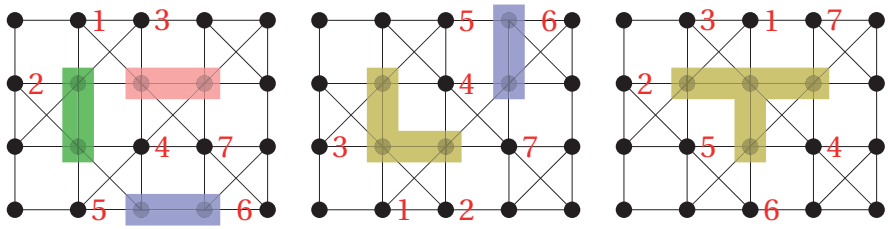
Many current and NISQ processors have geometrical connectivity constraints, that is, each qubit can only interact with a few neighbors. It is challenging or even impossible to directly perform existing flag-based quantum error correction without adding more operations and/or without losing fault tolerance. For example, the flag circuit which measures one weight-4 Z-stabilizer of the Steane code in Figure 6.1b cannot be directly executed on the Surface-17 topology (Figure 6.6a) but can be supported by the IBM-20 topology (Figure 6.6b). This is because qubit s needs to interact with 5 qubits but in Surface-17 each qubit has at most 4 neighbors. The flag circuit in Figure 6.1c can be performed on both processor topologies. However, a full round of error syndrome extraction requires all the stabilizer generators of the Steane code to be measured. The full syndrome extraction using only these two flag circuits (Figure 6.1b and Figure 6.1c) can be directly performed on the IBM-20 topology (e.g., a mapping in Figure 6.8a) but not on the Surface-17 topology.



(a) Steane-cl-L1 (b) Steane-c2-L1

Figure 6.7: Mapping of the Steane code onto the Surface-17 topology, where the qubits labelled with numbers are data qubits and the qubits in the colored blocks are ancillas. (a) The mapping using the two-ancilla flag-bridge circuits in Figures 6.1c and 6.4a in which only one stabilizer is measured at a time; (b) The mapping using the three-ancilla flag-bridge circuits in Figures 6.4b and 6.5a that measure one and two stabilizers, respectively.

As mentioned above, all the flag-bridge circuits shown in this chapter are used to measure Z stabilizers, similar circuits with the same ancillas can be derived for measuring X-stabilizers. Figures 6.7 and 6.8 show examples of mapping the Steane code error correction using the flag-bridge circuits onto the Surface-17 topology and the IBM-20



(a) Steane-cl-L2 (b) Steane-c2-L2 (c) Steane-c3-L2

Figure 6.8: Mapping of the Steane code onto the IBM-20 topology (a) using the flag-bridge circuits in Figures 6.1c and 6.4a; (b) using the flag-bridge circuits in Figures 6.4b and 6.5a; (c) using the four-ancilla flag-bridge circuit in Figure 6.5b to measure three stabilizers simultaneously.

topology, respectively.

In these mapping figures, the qubits in each red, blue, or green block are the ancillas in each flag-bridge circuit and they are used to measure the corresponding X(or Z)-stabilizer in the same color plaquette in Figure 6.2. The flag-bridge qubits in the yellow block are used to measure the X(or Z)-stabilizers in both red and green plaquettes. The flag-bridge qubits in the gray block measure the X(or Z)-stabilizers in all three plaquettes. The X and Z stabilizers are measured separately, more specifically, one first measures all the stabilizers in one type and then measures the other type. Furthermore, each of the flag-bridge circuits for the Steane code error correction need to be executed sequentially. On the Surface-17 topology, one can measure all the stabilizers of the Steane code one by one when using the mapping in Figure 6.7a. Maximally two stabilizers can be measured in parallel in this topology as shown in Figure 6.7b. In contrast, three $X(Z)$-stabilizers can be measured at the same time on the IBM-20 topology (Figure 6.8c).

The circuit characterization of one full round of syndrome extraction for the Steane code when using different mappings is shown in Table 6.1. This characterization includes the total number of ancilla qubits, the total number of operations and timesteps, and the number of f-CNOT and s-CNOT gates. For comparison, we also show these parameters of the distance-3 surface code (SC d=3). As shown in Table 6.1, the circuits which can measure more stabilizers simultaneously require fewer operations and fewer timesteps. Moreover, though the distance-3 surface code uses more ancilla qubits, it always needs fewer operations and fewer timesteps than the Steane code.

	# Ancillas	# Operations	# f-CNOTs	# s-CNOTs	# Timesteps
Steane-c1-L1	6	72	12	24	50
Steane-c1-L2	6	72	12	24	48
Steane-c2-L1	6	72	16	20	40
Steane-c2-L2	5	62	12	20	36
Steane-c3-L2	4	54	12	18	26
SC d=3	8	48	0	24	8

Table 6.1: Comparison of the quantum error correction circuits when different mappings are applied.

6.4.2. NUMERICS

We further compare different mapping circuits in terms of their fault tolerance, which is analyzed by numerical simulation under circuit level noise. For each point in the numerics, 10^6 iterations of a full QEC cycle have been run and confidence intervals at 99.9% are plotted. Moreover, NN decoders are used for this comparison since it has better performance than LUT decoders (see Figures 6.9a and 6.14). As shown in Figure 6.9, for the Steane code, the circuits that can measure more stabilizers in parallel have lower logical error rates, likely because they consist of fewer operations and require fewer timesteps. Moreover, when there are no idling errors ($p_I = 0$ in Figure 6.9a) or a small probability of idling errors ($p_I = 0.01p$ in Figure 6.9b), the Steane code can achieve similar performance to, or even outperform, the distance-3 surface code by parallelizing stabilizer measurements. This is because the circuit for the surface code error correction consists of more s-CNOT gates than the QEC circuits that can measure several stabilizers in parallel for the Steane code. When idling errors are significant, we observe that the circuit with fewer timesteps results in lower logical error rates (as shown in Figures 6.9c and 6.9d for $p_I = 0.1p$ and $p_I = p$ respectively).

6.5. OTHER APPLICATIONS OF THE FLAG-BRIDGE CIRCUITS

In this section, we foresee some possible applications of the flag-bridge circuits including both fault-tolerant quantum error correction and fault-tolerant quantum computation.

6.5.1. FLAG-BRIDGE QEC FOR THE FIVE-QUBIT CODE

Analogous to the flag circuits, the flag-bridge circuits can also be applied to other distance-3 error correction codes such as the $[\![8,3,3]\!]$, $[\![10,4,3]\!]$, $[\![11,5,3]\!]$, $[\![5,1,3]\!]$ codes, Hamming codes $\left[\!\left[2^r-1, 2^r-1-2r, 3\right]\!\right]$, etc. In this section, we consider the $[\![5,1,3]\!]$ code as an example. This code has four stabilizers, which are cyclic permutations of $XZZXI$. Figure 6.10 shows the flag-bridge circuits that can measure an $XZZX$ stabilizer fault-tolerantly. Each stabilizer of the 5-qubit code can be measured using these circuits up to data qubit permutation. Similar circuits using three ancillas to measure one stabilizer are also proposed in [114]. All these circuits have different connectivity requirements. By selecting and combining some of them, one can map the 5-qubit code error correction onto different qubit topologies. Figure 6.11 shows the mapping of the 5-qubit code to the Surface-17 processor topology using the two-ancilla flag-bridge circuits and the IBM Q Melbourne (IBM-16) processor topology using the three-ancilla flag-bridge circuits.

6.5.2. FLAG-BRIDGE CIRCUITS FOR FT COMPUTATION

The geometrical interaction constraint in near-term quantum processors also limits the fault-tolerant implementation of logical operations. For instance, a fault-tolerant CNOT gate in planar surface codes and color codes in principle can be implemented transversally in a 3D structure, that is, performing pair-wise CNOT gates between data qubits of the two lattices. However, this transversal CNOT is not realizable in near-term quantum technologies because of the local qubit connectivity limitation in a 2D architecture. Measurement-based protocols such as lattice surgery [18, 19] and code deforma-

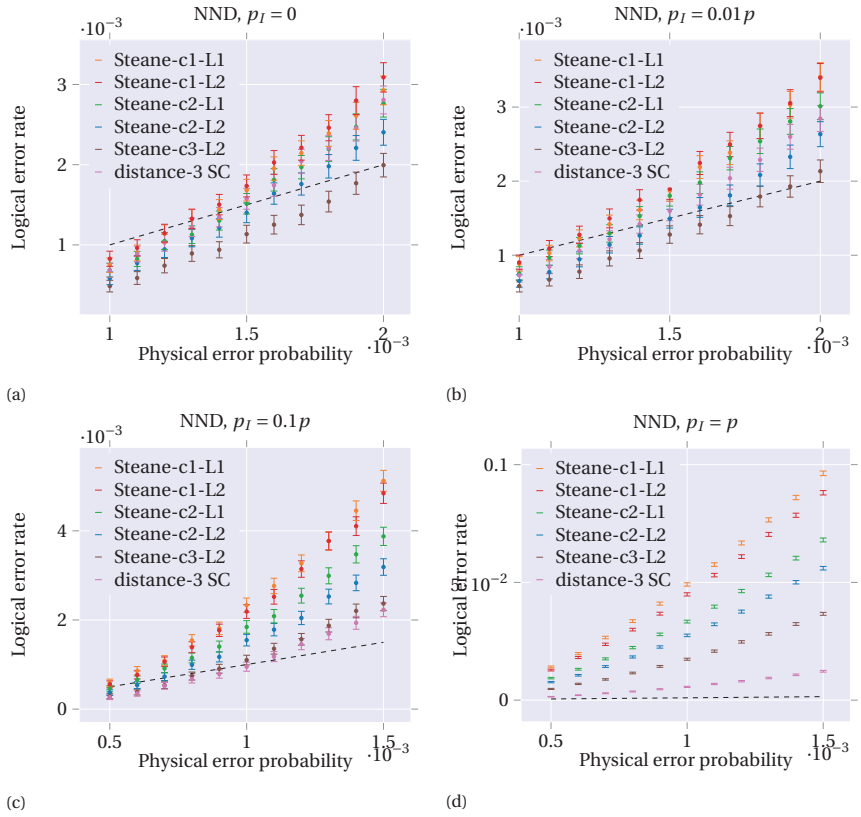


Figure 6.9: Numerical simulation of the Steane code error correction based on different flag-bridge circuits using neural network decoders. The circuit level noise ($p_1 = p_2 = p_M$) with ($p_I \neq 0$) or without idling errors ($p_I \neq 0$).

tion [20, 64] have been proposed to comply with the 2D local interaction constraint. Figures 6.12 and 6.13 show the qubit layouts for performing lattice-surgery-based operations on the distance-3 surface code and the distance-3 color code (the Steane code), respectively. The details of implementing logical operations by lattice surgery can be found in [18, 19].

As shown in Figure 6.12, the merge operations can be directly performed on a 2D grid topology. As mentioned previously, the stabilizer measurement of surface codes can be realized by only using the one-ancilla circuit similar to Figure 6.1a. However, one ancilla qubit (the circled one in Figure 6.12b) is used by two stabilizers from different lattices during the split operation. One may have to measure these two stabilizers sequentially, which leads to more timesteps and in turn may result in higher logical error rates. To preserve parallelism of the stabilizer measurement, we propose to use the qubit layout in Figure 6.12c. By using this layout, one can measure all the stabilizers in parallel when splitting lattices since they no longer share ancillas. One can also perform the merge operation by replacing the original syndrome extraction circuit using one ancilla

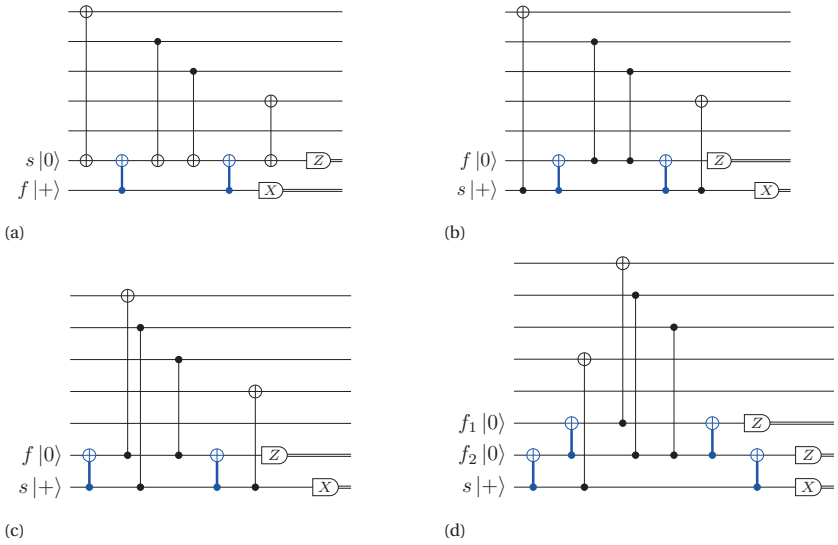


Figure 6.10: Fault-tolerant circuits for performing an $XZZX$-check: (a), (b), (c) using 2 ancillas but requiring different connectivity;(d) using three ancillas, similar circuits can be generated by re-distributing the s-CNOT gates for each weight-4 check to different ancillas as mentioned in Section 6.3.

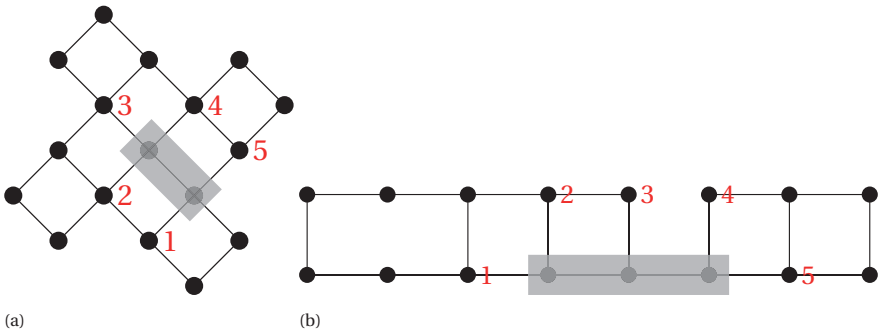


Figure 6.11: Map the 5-qubit code onto (a) the Surface-17 processor topology by combining the two-ancilla flag-bridge circuits in Figure 6.10 and (b) the IBM-16 processor topology using the three-ancilla circuit in Figure 6.10d.

with the proposed flag-bridge circuits using two ancillas (Figure 6.1c) where ancillas are connected by dash lines in Figure 6.12c . Similar mapping can be applied to other code-deformation-based operations on surface codes.

Furthermore, lattice-surgery-based operations for the Steane code in Figure 6.13a cannot be directly realized in a 2D grid topology. Similar to the mapping in Figure 6.7b, one can map these operations fault-tolerantly using the three-ancilla flag-bridge circuits as shown in Figure 6.13b. Compared to the distance-3 surface code, the Steane code can achieve Clifford gates transversally. Moreover, it requires fewer qubits for both FT error correction and FT computation, which may be preferable for demonstrating fault

tolerance in small experiments.

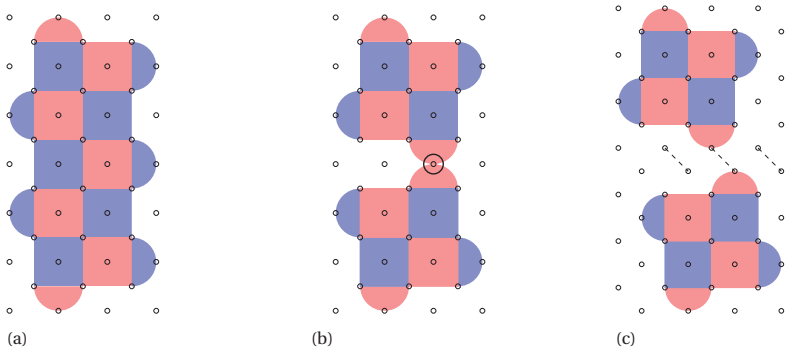


Figure 6.12: Mapping lattice surgery-based operations for the distance-3 surface code using flag-bridge circuits. Each red (blue) plaquette represents a weight-4 or weight-2 $X(Z)$-stabilizer. Data qubits are on the vertices and ancilla qubits are on the plaquettes. (a) and (b) Initial layouts for performing a merge and a split operation using lattice surgery, respectively. (c) The layout after mapping using the two-ancilla flag-bridge circuits.

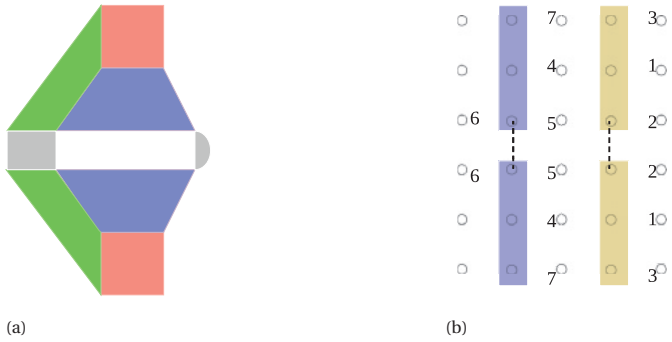


Figure 6.13: Mapping lattice surgery-based operations for the Steane code using flag-bridge circuits. (a) Initial layout, where the gray plaqeuttes are only one type stabilizers, depending on the joint measurement needs to be performed. Data qubits are on the vertices and ancilla qubits are on the plaquettes. (b) Mapping to a grid topology similar to Figure 6.7b.

6.6. DISCUSSION AND CONCLUSION

We have shown that flag circuits can be interpreted as replacing bare ancillas by encoded ancillas in an error detection code. Based on this formulation, we proposed a flag-bridge approach to perform fault-tolerant quantum error correction for distance-3 codes on connectivity-constrained near-term quantum processors with low overhead. Furthermore, we mapped the Steane code error correction onto two current qubit topologies using the flag-bridge circuits. The numerical simulation results show that QEC circuits that can measure more stabilizers in parallel achieve lower logical error rates, providing insights for fabricating processors with more connectivity. Moreover, we also showed the

flag-bridge circuits can be applied to the 5-qubit code and lattice-surgery-based operations for the surface codes and the Steane code. In addition, we have observed that the Steane code that use fewer qubits even outperforms the distance-3 surface code when qubit idling errors are negligible. Since the Steane code also allows transversal Clifford gates, it may be a better candidate than the distance-3 surface code for demonstrating fault tolerance in small experiments. However, because the numerics in this chapter were carried out with Pauli errors, it will be interesting to test these circuits using more realistic error models. Furthermore, the mapping procedure In this chapter was hand-optimized. Future work will focus on automating the fault-tolerant mapping of flag-bridge quantum error correction onto given processors. Besides, we also need to investigate the extendibility and scalability to higher distance codes and fault-tolerant computation.

6.7. IMPLEMENTATION OF LUT AND NN DECODERS

Based on the FT QEC procedure for distance-3 codes in Section 6.3, decoding is only needed when two rounds of syndrome extraction (SE) are performed (the first round has non-trivial syndromes or flags). If there is only non-trivial syndromes (no flags) in the first round, then the decoders will only decode using the measurement results in the second round. If there is any non-trivial flag in the first round, then the decoders will decode using these flags and the measurement results in the second round. For the measurement information in the second round, the simple LUT decoder only considers the results of syndrome qubits, which is enough for correcting all the errors caused by one fault. In contrast, the NN decoder also takes the flags of the second round into account. This means the NN decoder could potentially correct some errors caused by more faults, outperforming the LUT decoder.

As mentioned previously, we use a brute-force search to check the fault tolerance of flag-bridge circuits. After this search, all the errors from one single fault and the corresponding syndrome-flag (SF) string are collected. For FT flag-bridge circuits, these error-SF pairs can be directly used to design a LUT decoder. Two look-up tables need to be created. One is used for the case where only syndromes are observed in the first round of SE with a size 2^{m_s}, m_s is the total number of syndrome qubits in the QEC circuit $\mathscr{C} = \{c(g_1), c(g_2), \cdots, c(g_r)\}$. Note that if the same ancilla qubits are re-used in different c_g, they are still considered as different syndrome qubits, similarly for flag qubits. The other table is to decode for the case where flags are raised in the first round of SE, which has a size of $\sum_i 2^{m_{f_i}} 2^{m_s}$, m_{f_i} is the total number of flag qubits in c_{g_i}. The LUT decoder is designed to correct all single faults, but not to correct the most likely two faults correspond to measured syndromes. The performance of different flag-bridge circuits for the Steane code using LUT decoders is shown in Figure 6.14. As can be seen, the QEC circuits that can achieve more parallelism of stabilizer measurement have lower logical error rates.

The NN decoder: Decoding can be seen as a classification problem, that is, given the observed syndromes, the decoder identifies the error or the logical coset of the error that has occurred. It has been shown that neural networks are versatile tools for decoding topological quantum error correction codes [134, 36, 135, 37]. The inputs x_i for a neural network decoder are the syndromes (and flags for flag QEC). In this chapter, two rounds

of syndromes and flags will be collected when using the flag-bridge error correction for distance-3 codes. Therefore, the size of input layer will be $2 \times m$, m is the total number of syndrome and flag-bridge qubits. In this chapter, the outputs y_i are the suggested physical errors which can result in the given syndromes and flags. For a CSS code with n data qubits, the size of output layer is set to be $2 \times n$, which can describe whether a X or/and a Z error has occurred on each data qubit. The neural network will find an approximate function $f : x \rightarrow y$ to describe the input-output relation from the set of training data $\{(x_i, y_i)\}$. Note that for large-distance codes, it is more efficient to use logical errors as outputs and a simple decoder (e.g., LUT decoder) is required to generate the logical error information.

In this chapter, a simple NN decoder using the Tensorflow library [136] is developed to analyze the fault tolerance of different flag-bridge circuits. We use the 'sigmoid' activation function for the output layer and 10^5 syndrome-error pairs at physical error rate (PER) around 0.01 are sampled for each training, more details of the designed NN decoder are described in Table 6.2. Since the focus of this chapter is to evaluate the flag-bridge quantum error correction, we leave the performance and speed optimization of NN decoders for future work.

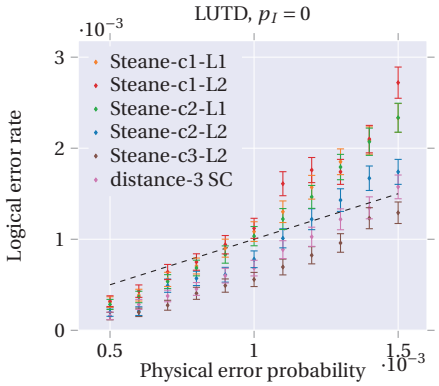


Figure 6.14: Performance of the LUT decoder for the Steane code under circuit level noise without idling errors.

Loss function	Hidden layers	Activation function		Optimizer	Learning rate	Batch size	PER	Samples
		Output layer	hidden layer					
cross-entropy	3	sigmoid	ReLU (tanh)	Adam (Nadam)	0.002	50	~ 0.01	10^5

Table 6.2: The implementation details of the NN decoder.

7

Mapping of Quantum Circuits onto NISQ Superconducting Processors

Several quantum processors consisting of a few tens of noisy qubits have already been constructed, and are called Noisy Intermediate-Scale Quantum (NISQ) devices. Their low number of qubits precludes the use of quantum error correction procedures, and so only small size quantum algorithms can be successfully run. These quantum algorithms need to be compiled to respect the constraints imposed by the quantum processor; this is known as the mapping problem. This mapping will result in an increase of the number of gates and of the circuit depth, decreasing the algorithm's success rate.

In this chapter, we present a mapper called Qmap that makes quantum circuits executable on the Surface-17 processor. It takes into account not only the elementary gate set and qubit connectivity constraints but also the restrictions imposed by the use of shared classical control, which have not been considered so far. Qmap is embedded in the OpenQL compiler and uses a configuration file where the processor's characteristics are described and that makes it capable of targeting different quantum processors. To show this flexibility and evaluate its performance, we map 56 quantum benchmarks on two different superconducting quantum processors, the Surface-17 (17 qubits) and the IBM Q Tokyo (20 qubits), while using different routing strategies.

This chapter is based on L. Lao, D. M. Manzano, J. van Someren, I. Ashraf, C. G. Almudever, *Mapping of quantum circuits onto NISQ superconducting processors,* arXiv:1908.04226 (2019).

7.1. INTRODUCTION

Quantum computers promise to solve a certain set of complex problems that are intractable by the best known classical algorithms, the most famous example being the factorization of large numbers using Shor's algorithm [137]. However, a fault-tolerant (FT) large-scale quantum computer with thousands or even millions of qubits will be required to solve such a kind of problem [15, 138].

Quantum computing is still far away from that as it is now just entering the Noisy Intermediate-Scale Quantum (NISQ) era [21]. This refers to exploiting quantum processors consisting of only 50 to a few hundreds of noisy qubits - i.e qubits with a relatively short coherence time and faulty gates [3, 14]. Due to the limited number of qubits, hardly or no quantum error correction (QEC) will be used in the next coming years posing a limitation on the size of the quantum applications that will be successfully run on NISQ processors. Nevertheless, these processors will still be useful to explore quantum physics, and implement small quantum algorithms that will hopefully demonstrate quantum advantage [21]. For running near-term quantum applications on noisy quantum devices, it is thus crucial to minimize their size in terms of circuit width (number of qubits), number of gates, and circuit depth (number of cycles or steps) [71, 139].

In addition, these quantum applications have to be adapted to the hardware constraints imposed by current quantum processors. The main constraints include:

- Elementary gate set: Generally only a limited set of quantum gates that can be realized with relatively high fidelity will be predefined on a quantum device. Each quantum technology may support a specific universal set of single-qubit and two-qubit gates. For instance, some superconducting quantum technologies have CZ as an elementary two-qubit gate [140, 141].

- Qubit connectivity: quantum technologies such as superconducting qubits [14, 16, 45, 69] and quantum dots [17, 46] arrange their qubits in 2D architectures with *nearest-neighbor* (NN) interactions. This means that only neighboring qubits can interact or in other words, qubits are required to be adjacent for performing a two-qubit gate. In other technologies such as trapped-ion qubits, they are fully connected and allow all-to-all interactions [12].

- Classical control: classical electronics are required for controlling and operating the qubits. Using a dedicated instrument per qubit is not scalable and very expensive approach. Therefore, shared control is required especially when building scalable quantum processors. For instance, a single Arbitrary Waveform Generator (AWG) is used for operating on a group of qubits and several qubits are measured through the same feedline [142, 143]. This limits the possible parallelism of quantum operations, leading to longer latencies and a larger circuit depth.

All these constraints may vary between different qubit implementations and even within the same quantum technology. In order to meet them, a mapping procedure is required to transform a hardware-agnostic quantum circuit into a hardware-aware version that can be run on a given quantum processor. Mapping will: i) map virtual qubits (qubits in the circuit) to hardware qubits (physical qubits in the processor), ii) route qubits to move non-adjacent qubits to neighboring positions when they need to interact. For this purpose, the path that the qubits will follow needs to be determined and movement operations such as shuttling in trapped ion and Si-spin quantum processors [47, 46], and SWAPs in superconducting quantum processors will be inserted accordingly. Note that routing will increase the number of operations as well as the circuit depth. iii) Schedule the operations respecting not only the dependencies between them but also the classical control constraints. In addition, gates will be decomposed to elementary gates and the circuit will be optimized at different stages of the compilation process. For NISQ processors it is key to minimize the mapping overhead such that the resulting circuit still has a high reliability and success rate. Note that the higher the number of gates and/or the circuit depth, the higher the failure rate of computation and thus the lower the reliability of the circuit.

Different solutions have been proposed to map quantum circuits onto current and NISQ processors. [72, 73, 74, 75, 76, 77, 78, 79, 144] map quantum algorithms onto processors with a 2D grid structure. [80, 82, 81, 145, 146, 147, 148, 149] and [83, 130, 149] propose mapping algorithms targeting IBM and Rigetti processors, respectively. Most of the works done so far focus on quantum processors that do not have scalable architectures and mainly consider the connectivity constraint and the elementary gate set. Furthermore, they do not consider other hardware information such as gate duration and shared classical control that should be also taken into account to make quantum applications executable [39, 143].

This paper is the first to map quantum circuits to the Surface-17 processor, a scalable processor with a surface code architecture. It presents a mapper called **Qmap** that takes into account all three types of constraints: the elementary set of gates with corresponding duration, qubit connectivity, and control electronics. Qmap is composed of several modules, including initial placement of qubits, routing of qubits, and gate scheduling, together with decomposition and optimization steps. It is embedded in the OpenQL compiler [1] [150] and quantum hardware information is described in a configuration file that is used by all compiler passes, providing the flexibility to be applied to different quantum simulators and underlying processors. The mapper takes as input a quantum program written in OpenQL (C++ or Python), maps and optimizes the corresponding quantum circuit for the given quantum platform and generates executable low-level QASM-like code [151]. The mapper is used not only for mapping quantum circuits to the Surface-17 processor but also to the IBM Q Tokyo chip [14] to show its universality. Several benchmarks that differ in number of qubits and gates are evaluated. We analyze the overhead caused by mapping in terms of the number of extra gates and circuit depth/latency when using different routing strategies. The main contributions of

[1] OpenQL is an open-source quantum programming language and compiler developed by the Quantum Computer Architecture Lab/QuTech, Delft University of Technology. The Qmap mapper will be included in the next OpenQL release.

this paper are the following:

- We have developed a mapper (Qmap) for a scalable superconducting quantum processor such as the Surface-17. The mapper considers not only common processor constraints such as the choice of the elementary gate set and the qubit connectivity but also gate execution time (gate duration) and classical control constraints resulting from using control electronics that are shared among qubits.
- With the goal of supporting several quantum processors, our mapper has been embedded in the OpenQL compiler. It can target different quantum chips by using a configuration file in which the constraints of the processor are described. This flexibility allows performing a comparative analysis between them and giving some directions for building future quantum machines. We compile 56 benchmarks taken from RevLib [152] and QLib [102] onto two quantum processors, the Surface-17 and the IBM Q Tokyo processors.
- The developed mapper supports different routing strategies. Three of them are used and evaluated in this chapter (Trivial, MinPath and MinRC). After mapping using the best router, the circuit latency can increase up to 260% and the overhead in the number of gates can be as high as 78.1% for the Surface-17 processor.
- Our mapper uses not only SWAP operations (3 CNOTs) for moving qubits but also MOVE operations (2 CNOTs) when possible. The use of MOVEs helps to reduce the number of gates and the circuit latency up to 38.9% and 29%, respectively.
- A numerical analysis on how the mapping affects the reliability of some small quantum circuits when mapped to the Surface-17 chip is also presented. They show a fidelity decrease that ranges from 1.8% to 13.8%.

The rest of this paper is organized as follows. We first describe all the hardware parameters that will be considered in this chapter in Section 7.2. Then the proposed mapping procedure and corresponding routing algorithms are introduced in Section 7.3 and evaluated in Section 7.4. We summarize related works in Section 7.5. Finally, Section 7.6 concludes the paper and discusses future work.

7.2. QUANTUM HARDWARE CONSTRAINTS

In this section, the hardware constraints of the superconducting Surface-17 and the IBM Q Tokyo quantum processors will be briefly introduced, including the primitive gates that can be directly performed, the topology of the processor which limits interactions between qubits, and the constraints caused by the classical control electronics which impose extra limitations on the parallelism of the operations.

7.2.1. ELEMENTARY GATE SET

In order to run any quantum circuit, a universal set of operations needs to be implemented. In superconducting quantum processors, these operations commonly are measurement, single-qubit rotations, and multi-qubit gates.

Surface-17 processor: In principle, any kind of single-qubit rotation can be performed on the Surface-17 processor. However, an uncountable number of gates cannot be predefined. In this chapter, we will limit single qubit gates to X and Y rotations (easier to implement), and more specifically ± 45, ± 90 and ± 180 degrees will be used in our decompositions. The primitive two-qubit gate in transmon is the conditional-phase (CZ) gate. Table 7.1 shows the gate duration (gate execution time) of single-qubit gates, CZ gate and measurement (in the Z basis) [24]. After mapping, the output circuit will only contain operations that belong to this elementary gate set. The decomposition for $Z, H, S, S^\dagger, T, T^\dagger$, CNOT, SWAP and MOVE gates into the elementary gates shown in Table 7.1 can be found in Figure 7.1.

Table 7.1: The gate duration in cycles (each cycle represent 20 nanoseconds) of the elementary gates in the Surface-17 processor.

Gate type	Duration
$R_X(\pm 45, \pm 90, \pm 180)$	1 cycle
$R_Y(\pm 45, \pm 90, \pm 180)$	1 cycle
CZ	2 cycles
M_Z	15 cycles

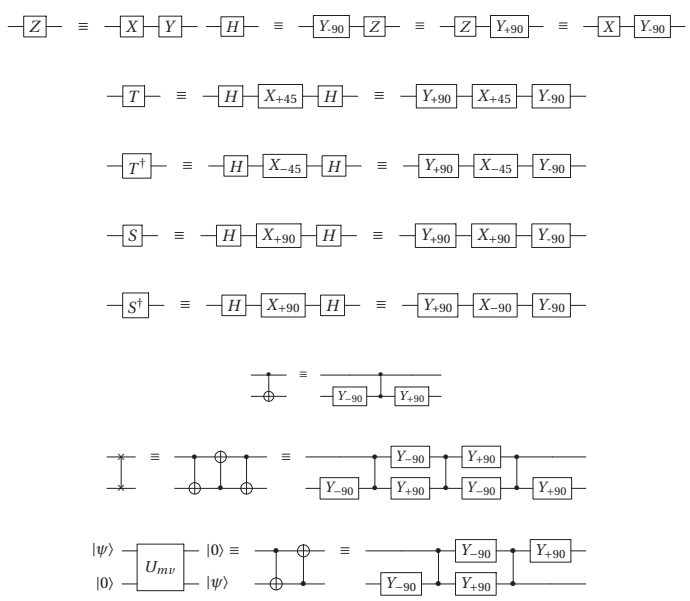
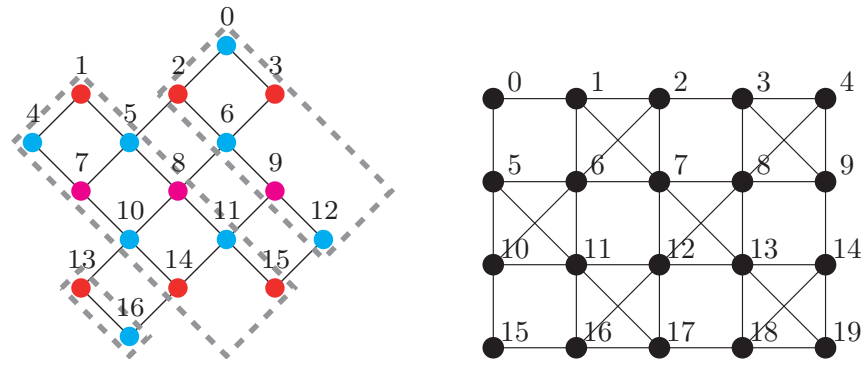


Figure 7.1: Gate decomposition into primitives (Table 7.1) supported in the superconducting SC-17 processor. U_{mv} is the MOVE operation.

IBM Q Tokyo (IBM-20): The elementary gates supported by the IBM Q processors are any single-qubit rotation $U(\theta, \phi, \lambda) = R_Z(\phi)R_Y(\theta)R_Z(\lambda)$ and the conditional-NOT (CNOT) gate. This means that the gate set { Pauli, $H, S, S^\dagger, T, T^\dagger$, CNOT} can be directly

supported without further decomposition. In this chapter, we do not take the gate duration of the IBM Q Tokyo processor into account since the authors did not find the duration of two-qubit gates of this device by the time of writing this thesis.

7.2.2. PROCESSOR TOPOLOGY



(a) (b)

Figure 7.2: (a) Schematic of the realization of the Surface-17 processor and (b) the topology of the IBM Q Tokyo processor

Figure 7.2 shows the topology of the Surface-17 and the IBM Q Tokyo processors, where nodes represent the qubits and edges represent the 'connections' (resonators) between them. Two-qubit gates can only be performed between connected qubits, i.e., *nearest-neighbouring* qubits. This implies that qubits that have to interact but are not placed in neighbouring positions will need to be moved to be adjacent. Quantum states in superconducting technology are usually moved using SWAP gates. A SWAP gate is implemented by three CNOTs that in the case of the Surface-17 processor need to be further decomposed into CZ and R_Y gates as shown in Figure 7.1. In this chapter, we also consider the use of a MOVE operation which only requires two CNOTs (see Figure 7.1). Note that a MOVE operation requires that the 'destination' qubit where the quantum state needs to be moved to, is in the $|0\rangle$ state. As mentioned, moving qubits results in an overhead in terms of number of operations and circuit depth, which in turn will decrease the circuit reliability.

7.2.3. CLASSICAL CONTROL CONSTRAINTS

In principle, any qubit in a processor can be operated individually and then any combination of single-qubit and two-qubit operations can be performed in parallel. However, scalable quantum processors use classical control electronics with channels that are shared among several qubits. Here we will describe how the classical control electronics used in the Surface-17 processor affect the parallelism of operations. The classical control constraints for the IBM Q Tokyo processor were not found by the authors and therefore they will not be considered in this chapter.

Single-qubit gates: Single-qubit gates on transmons are performed by using microwave pulses. In Surface-17, these pulses are applied at a few fixed specific frequencies to ensure scalability and precise control. The three frequencies used in Surface-17 are shown in Figure 7.2a: single-qubit gates on red, blue and pink colored qubits are performed at frequencies f_1, f_2, and f_3, respectively [39]. In this chapter, we assume that same-frequency qubits are operated by the same microwave source or arbitrary waveform generator and a vector switch matrix (VSM) is used for distributing the control pulses to the corresponding qubits [142].

The consequence of this is that one can perform the same single-qubit gate on all or some of the qubits that share a frequency, but one cannot perform different single-qubit gates at the same time on these qubits (as these would require other pulses to be generated). For instance, an X gate can be performed simultaneously on any of the pink qubits (7, 8 and 9) but not an X and a Y operation.

Measurement: Measuring the qubits is done by using feedlines each of which is coupled to multiple qubits [39]. In Figure 7.2a, qubits in the same dashed rectangle are using the same feedline, e.g., qubits 13 and 16 will be measured through the same feedline. Because measurement takes several steps in sequence, measurement of a qubit cannot start when another qubit coupled to the same feedline is being measured, but any combination of qubits that are coupled to the same feedline can be measured simultaneously at a given time. For instance, qubits 13 and 16 can be measured at time t_0, but it is not possible to start measuring qubit 13 at time t_0 and then measure qubit 16 at time t_1 if the previous measurement has not finished.

Two-qubit gates: As mentioned, in the processor of Figure 7.2a each qubit belongs to one of three frequency groups $f_1 > f_2 > f_3$, colored red, blue and pink, respectively; links between neighboring qubits are either between qubits from f_1 and f_2, or between qubits from f_2 and f_3, i.e. between a higher frequency qubit and a next lower one. In between additional frequencies are defined: $f_1 > f_1^{int} > f_2 > f_2^{park} > f_2^{int} > f_3 > f_3^{park}$ (see the frequency arrangement and the example interactions presented in Figure 5 of [39]); each qubit can be individually driven with one of the frequencies of its group. A CZ gate between two neighboring qubits is realized by lowering the frequency of the higher frequency qubit near to the frequency of the lower one. For instance, a CZ gate between qubits 3 and 0 is performed by detuning qubit 3 from f_1 to f_1^{int}, which is near to f_2, the frequency of qubit 0. However, CZ gates will occur between any two neighboring qubits which have close frequencies and share a connection, e.g. between qubits 3 and 6 in the given example. To avoid this, the qubits that are not involved in the CZ gate must be kept out of the way. In this example, q6 is detuned to a lower *parking frequency*, f_2^{park}. Note that qubits in parking frequencies cannot engage in any two-qubit or single-qubit gate. In addition, qubit 2 must stay at f_1 when qubits 3 and 0 perform a CZ, to avoid interaction between qubits 2 and 0. This example shows that the implementation of two-qubit gates poses some limitations on gate parallelism.

The hardware characteristics described in this section are included in a configuration file (in json format) that is used by all modules of the mapper.

7.3. MAPPING QUANTUM ALGORITHMS: THE QMAP MAPPER

Mapping means to transform the original quantum circuit that describes the quantum algorithm and is hardware-agnostic to an equivalent one that can be executed on the target quantum processor. To this purpose, the mapping process has to be aware of the constraints imposed by the physical implementation of the quantum processor. These include the set of elementary gates that is supported, the allowed qubit interactions that are determined by the processor topology, and the limited concurrency of multi-gate execution because of classical control constraints. Because of mapping, the number of operations that are required to implement the given algorithm as well as the circuit depth are likely to increase, decreasing the reliability of the algorithm. An efficient mapping is then key, especially in NISQ processors where noise sets a limit on the maximum size of a computation that can be run successfully.

7.3.1. OVERVIEW OF THE QMAP MAPPER

The **Qmap** mapper developed in this chapter is embedded in the OpenQL compiler [150]. We show its flow in Figure 7.3. Its input is a quantum circuit written in OpenQL (C++ or Python). The OpenQL compiler reads and parses it to a QASM-level intermediate representation. Qmap then performs the mapping and optimization of the quantum circuit based on the information provided in a configuration file that includes the processor topology (connectivity and number of qubits), its elementary gate set, gate decomposition rules, the duration of each gate, and the classical control constraints. After mapping, QASM-like code is generated. Currently, the OpenQL compiler is capable of generating cQASM [151] that can be executed on our QX simulator [153] as well as eQASM [107], a QASM-like executable code that can target the Surface-17 processor. The generation of other QASM-like languages will be part of future extensions of the OpenQL compiler.

Note that as the characteristics of the quantum processor are described in a configuration file that is provided to the mapper, Qmap can easily target different quantum devices just by providing it with different configuration files with appropriate parameters.

The modules of Qmap will be discussed in the next sections. We refer to the qubits in the quantum circuit as virtual qubits (others call them program qubits or logical qubits). These need to be mapped to the qubits in the quantum processor called physical, real or hardware qubits or locations.

7.3.2. INITIAL PLACEMENT

Qubits are preferably placed initially such that highly interacting qubits are placed next to each other. Qmap tries to find an initial placement that minimizes the number of qubit movements by using the Integer Linear Programming (ILP) algorithm presented in [115]. Similar to the placement approaches in [76, 86, 99], the initial placement problem is formulated as a quadratic assignment problem (QAP) with the communication overhead between qubits modeled by their distance minus 1. Such an initial placement implementation can only solve small-scale problems in reasonable time. Even though for near-term implementations these numbers largely suffice, for large-scale circuits, one can either partition a large circuit into several smaller ones or apply heuristic algorithms to efficiently solve these mapping models [72, 73, 74, 76, 84, 85, 87]. Other works solve

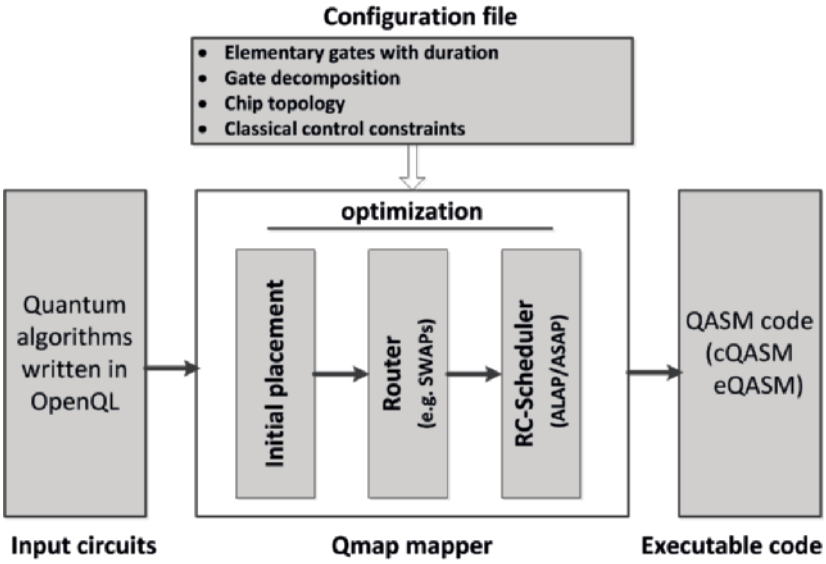
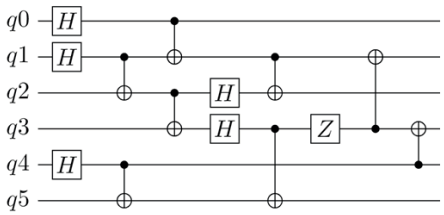


Figure 7.3: Overview of the Qmap mapper embedded in the OpenQL compiler.



V	q0	q1	q2	q3	q4	q5
P	q12	q15	q11	q8	q1	q5

(a)

```
h q[0]
h q[1]
h q[4]
cnot q[1],q[2]
cnot q[4],q[5]
cnot q[0],q[1]
cnot q[2],q[3]
h q[2]
h q[3]
cnot q[1],q[2]
cnot q[3],q[5]
z q[3]
cnot q[3],q[1]
cnot q[4],q[3]
```

(b)

Figure 7.4: An example circuit consisting of 6 qubits and 15 gates. (a) Its circuit description (top) and its virtual to physical qubits mapping (bottom) for the Surface-17 processor after initial placement; (b) Its cQASM representation before routing and scheduling.

this initial placement problem by using a Satisfiability Modulo Theories (SMT) solver [154].

An example circuit, its virtual to physical qubits mapping found by the initial placement module and the cQASM code before routing and scheduling are shown in Figure 7.4.

7.3.3. QUBIT ROUTER

It is unlikely to find an initial placement in which all interactions between the qubits are satisfied. That is, not all the qubits that will perform a two-qubit gate can be placed in neighboring positions. Therefore, they will have to be moved during computation. For instance, based on the initial placement of qubits proposed in Figure 7.4a, the first 6 CNOTs of the circuit can be performed directly as qubits are NN, but the last 2 CNOTs will require qubits to be routed to adjacent positions. Routing refers to the task of finding a series of movements that enables the execution of two-qubit gates on a given processor topology with low communication overhead. To do so, several routing paths are explored and one is selected based on various optimization criteria such as the number of added movement operations, increase of circuit depth, or decrease of circuit reliability [80, 81, 82, 145, 146, 147, 148, 130, 83, 155, 154]. Then, the corresponding movement operations are inserted.

In this chapter next to and after the ILP-based initial placement, a heuristic algorithm is used to perform this routing task. It is a graph-based heuristic of which the objective is to achieve the shortest circuit latency and therefore the highest instruction-level parallelism. Algorithm 2 shows the pseudo code of our routing algorithm; it finds all two-qubit gates in which qubits are not nearest-neighbors and inserts the required movement operations to make them adjacent. As mentioned in Section 7.2.2 we use SWAPs as well as MOVE operations for moving qubits. The algorithm works as follows:

1. From the QASM representation of the quantum circuit a Quantum Operation Dependency Graph (QODG) $G(V_G, E_G)$ is constructed, in which each operation is denoted by a node $v_i \in V_G$, and the data dependency between two operations v_i and v_j is represented by a directed edge $e(v_i, v_j) \in E_G$ with weight w_i that represents the duration of operation v_i. Pseudo source and sink nodes are added to the start and end to simplify starting and stopping iteration over the graph. The QODG of the circuit in Figure 7.4a is shown in Figure 7.5a.

2. The router algorithm starts by mapping the pseudo source node and then selecting all available operations from the input QODG, that is, the operations that do not depend on any not yet mapped operation. As long as among these are single-qubit gates or two-qubit gates with qubits that are NN, these are mapped first and a new set of available operations is computed. Mapping a (NN) gate implies replacing virtual qubit operands by their physical counterparts according to the table similar to the one shown in Figure 7.4a and decomposing it to its primitives when the configuration specifies so. After that, only non-NN two-qubit gates remain in the available set. The router, looking ahead to all not yet mapped operations of the circuit, selects from this availability set the ones which are most critical in the remaining dependency graph since they have the highest likelihood to extend the circuit when mapped in an inefficient way or when delayed. When there are several of these equally critical, it takes of these the first in the input circuit to map. After mapping it, it recomputes the set of available operations and runs the algorithm until there are no available operations anymore.

3. When mapping a non-NN two-qubit gate, all shortest paths between the qubits involved in this gate are considered. During Qmap initialization time, the distance

(i.e. the length of the shortest path) between each pair of qubits has been computed using the Floyd-Warshall algorithm. Finding all shortest paths between a pair of qubits at mapping-time is done by a breadth-first search (BFS), selecting only path extensions which decrease the distance between the qubits. For each shortest path, several movement sets are computed. Each movement set consists of a sequence of movements that brings the two qubits to adjacent positions. That is, qubits can meet in any neighboring position within the path. Note that all movement sets would lead to adding an equal minimum number of movements to the circuit. To choose the best movement set, several strategies can be used that differ in how the movement set is selected and what constraints are considered. In this chapter, we consider three to be compared and evaluated:

MinRCRouter: As shown in Algorithm 3, this routing strategy evaluates all movement sets by looking back to the previously mapped operations and interleaving each set of movements with those operations using an as-soon-as-possible (ASAP) scheduling policy. Then, it selects the one(s) which minimally extend(s) the circuit depth or latency. When there are multiple minimal sets, a random one is taken. The scheduling in this strategy takes gate duration and the classical control resource constraints into account, the latter limiting instruction-level parallelism. Its aim is to minimize the extension of the circuit latency caused by the addition of the movements by maximizing instruction-level parallelism within the constraints of the system.

MinPatouter: It just randomly selects one of the movement sets that are generated as described above, i.e. does not evaluate them for their extension of the circuit latency or depth.

TrivialRouter: The gates in the circuit are mapped in the order as they appear in the circuit, i.e. by-passing the QODG. Then, when there is a non-NN two-qubit gate, only the first shortest path that is found, is taken. In addition, a single movement set is generated for it; the one moving the control qubit until it is near to the target. From the movement set, only SWAPs are generated, not MOVEs.

After the movement set selection, the SWAP/MOVE operations are scheduled into the output circuit and the set of available gates and the map of virtual to physical qubits are updated.

Algorithm 2 Routing algorithm

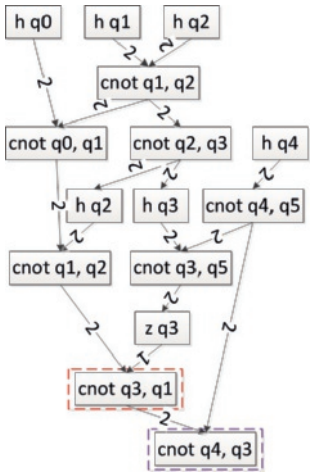
Require: Non-routed circuit, VP-map M, JSON file
Ensure: Routed circuit
1: Generate QODG $G(V_G, E_G)$
2: $V_m \leftarrow$ Unique pseudo source node
3: $V_{av} \leftarrow$ All available gates in $G(V_G - V_m, E_G)$
4: **while** $V_{av} \neq \varnothing$ **do**
5: $V_{nn} \leftarrow$ All single-qubit and NN two-qubit gates in V_{av}
6: **if** $V_{nn} \neq \varnothing$ **then**
7: Select $v \in V_{nn}$ arbitrarily
8: **else**
9: $V_c \leftarrow$ Most-critical gates $\subset V_{av}$ in $G(V_G - V_m, E_G)$
10: Select $v \in V_c$ which is first in the circuit
11: Insert movement(s) for v
12: Update M
13: Map v according to M
14: Add v to V_m
15: $V_{av} \leftarrow$ All available gates in $G(V_G - V_m, E_G)$

Algorithm 3 Movement insertion algorithm

Require: QODG $G(V_G, E_G)$, gate v, VP-map M, JSON file
Ensure: The set of movements for v
1: $P \leftarrow$ All shortest paths for v
2: $MV_P \leftarrow$ All possible sets of movements based on P
3: **for** mv_j in MV_P **do**
4: Interleave mv_j with previous gates (looking back)
5: $T_{mv_j} \leftarrow$ circuit's latency extension by mv_j
6: **if** $T_{mv_i} = min(\bigcup_j T_{mv_j})$ **then**
7: Select mv_i as the set of movements, picking a random minimum one when there are more

7.3.4. RC-SCHEDULER

After routing, the circuit adheres to the processor topology constraint for two-qubit interactions, and has been scheduled in an as-soon-as-possible (ASAP) way, taking the resource constraints into account only in the case of the MinRC router. The RC-scheduler reschedules the routed circuit to achieve the shortest circuit latency and the highest instruction-level parallelism. It does this in an as-late-as-possible (ALAP) way to minimize the required life-time and thus the decoherence error of each qubit, while taking the resource constraints into account. The resource constraints encode the control concurrency limitations together with the duration of the individual gates.



(a)

```
y90 q[15]
{ x q[15] | ym90 q[11] }
cz q[15],q[11]
wait 1
y90 q[12]
x q[12]
{ y90 q[11] | ym90 q[15] | ym90 q[8] }
{ cz q[12],q[15] | cz q[11],q[8] | y90 q[1] }
{ ym90 q[5] | x q[1] }
{ x q[11] | cz q[1],q[5] }
{ y90 q[15] | y q[11] }
{ x q[8] | cz q[1],q[5] | cz q[15],q[11] }
y q[8]
{ cz q[15],q[11] | cz q[8],q[5] }
wait 1
{ y90 q[5] | y90 q[11] | ym90 q[1] | ym90 q[15] }
{ cz q[5],q[1] | cz q[11],q[15] }
 x q[8]
{ ym90 q[5] | ym90 q[11] | y90 q[1] | y q[8] }
{ cz q[8],q[11] | cz q[1],q[5] }
wait 1
{ y90 q[5] | y90 q[15] | ym90 q[8] }
{ cz q[5],q[8] | cz q[15],q[11] }
wait 1
{ y90 q[11] | y90 q[8] }
```

(b)

Figure 7.5: (a) The QODG of the circuit in Figure 7.4a. The red and purple CNOTs mean that the qubits are not NN. (b) The cQASM code of the mapped circuit, where the CZ gates in bold are already nearest neighboring. Movement operations (added two-qubit gates are in yellow) are inserted to perform the CZ gates in red and purple.

7.3.5. DECOMPOSITION AND OPTIMIZATION

Starting from a quantum circuit described in cQASM format (see Figure 7.4), the circuit is also decomposed during mapping into one which only contains the *elementary gates* specified in the **configuration file** (json file), on top of adherence to the other constraints. In addition, it is optimized to reduce the number of operations, e.g., two consecutive X gates can cancel each other out.

The decomposition and optimization can be done at every step of the mapping procedure, i.e. before, during, and after routing. Qmap reduces sequences of single qubit operations to their minimally required sequence both before and after routing. The implementation of the QODG represents the commutability of all gates with disjoint qubit operands but also of the known two-qubit operations CNOT and CZ with overlapping operands, and optimizes their order, both during routing and during RC-scheduling. These optimizations are not performed in the TrivialRouter. Whether gate decomposition is to be applied at a mapping step is specified in the configuration (json) file.

The cQASM code generated after the Qmap mapper is shown in Figure 7.5b.

7.4. QMAP EVALUATION

In this section, we evaluate the *Qmap* by mapping a set of benchmarks from RevLib [152] and QLib [102] on two superconducting processors, namely, the transmon processor with a distance-3 surface-code topology (Surface-17) [39] and the IBM Q Tokyo (IBM-20) processor [14]. These two processors have different elementary gate sets, processor

topologies, and hardware constraints as described in Section 7.2. Specifically, for the Surface-17 processor, the elementary gates with their real gate duration, the topology and the electronic control constraints are considered. For the IBM-20 processor, only the elementary gates without considering their duration and the qubit topology are considered. All mapping experiments are executed on a sever with 2 Intel Xeon E5-2683 CPUs (56 logical cores) and 377GB memory. The Operating System is CentOS 7.5 with Linux kernel version of 3.10 and GCC version of 4.8.5.

7.4.1. BENCHMARKS

The circuit characteristics of the used benchmarks are shown in Table 7.2. All circuits have been decomposed into ones which only consist of gates from the universal set {Pauli, $S, S^\dagger, T, T^\dagger, H$, CNOT}. In these benchmarks, the number of qubits varies from 3 to 16, the number of gates goes from 5 to 64283, and the percentage of CNOT gates varies from 2.8% to 100%. Moreover, the minimum circuit depth and the minimum circuit latency are also included, ranging from 2 to 35572 time-steps and from 5 to 12256 cycles (using the gate duration of Surface-17), respectively. Note that these numbers are meant to characterize the algorithms before being mapped to the quantum processor and therefore are obtained without considering any hardware constraint.

The latter two parameters will be also used as a metrics to evaluate our Qmap mapper. They can be defined as follows:

Circuit depth is the length of the circuit. It is equivalent to the total number of time-steps for executing the circuit assuming each of the gates takes one time-step.

Circuit latency refers to the execution time of the circuit considering the real gate duration. Latency and gate duration are expressed in cycles. In this chapter, we assume that a cycle takes 20 nanoseconds.

In addition, other parameters after mapping the benchmarks to the two quantum processors are provided, such as the total number of gates and two-qubit gates, the number of inserted SWAP and MOVE operations, and the time the mapping process takes.

7.4.2. MAPPING RESULTS

Figure 7.6 and Figure 7.7 show the overhead of mapping the benchmarks in Table 7.2 to the superconducting Surface-17 processor and IBM-20 processor respectively using the three different routers: Trivial, MinPath, and MinRC router. More details of the mapping results for the Surface-17 and IBM-20 can be found in Table 7.4 and Table 7.5, respectively. The benchmarks in all the Figures in this chapter are shown in their appearing order (from top to bottom and left to right) in Table 7.2. We take the results of the mapping with the Trivial router as a baseline. In this case, a naive initial placement is used in which qubits are just placed in order, no optimization is made, and only SWAP operations are inserted. The mapping results for the MinPath and MinRC routers use the ILP-based initial placement. As we mentioned, this method can only solve small scale problems (small circuits) in a reasonable time. Note that the qubit initial placement is an NP-hard problem. In this chapter, the mapper is set to only find an initial placement for the first ten two-qubit gates in any given circuit and computation time is limited to 10 minutes. In addition, when using the MinPath and MinRC routers, circuit optimizations are enabled and both SWAP and MOVE gates are inserted. The mapping procedure is

Table 7.2: The characteristics of the input benchmarks including the number of qubits, the total number of gates, the number of two-qubit gates (CNOTs), its circuit depth (DP) and its circuit latency (LC) in cycles (20 ns per cycle).

Benchmarks	Qubits	Gates	CNOTs	DP	LC
alu_bdd_288	7	84	38	48	169
alu_v0_27	5	36	17	21	72
benstein_vazirani	16	35	1	5	40
4gt12_v1_89	6	228	100	130	448
4gt4_v0_72	6	258	113	137	478
4mod5_bdd_287	7	70	31	40	140
cm42a_207	14	1776	771	940	3249
cnt3_5_180	16	485	215	207	729
cuccaroAdder_1b	4	73	17	25	58
cuccaroMultiply	6	176	32	55	133
decod24_bdd_294	6	73	32	40	143
decod24_enable	6	338	149	190	669
graycode6_47	6	5	5	5	20
ham3_102	3	20	11	11	41
miller_11	3	50	23	29	105
mini_alu_167	5	288	126	162	564
mod5adder_127	6	555	239	302	1048
mod8_10_177	6	440	196	248	872
one_two_three	5	70	32	40	141
rd32_v0_66	4	34	16	18	66
rd53_311	13	275	124	124	441
rd73_140	10	230	104	92	330
rd84_142	15	343	154	110	394
sf_274	6	781	336	436	1516
shor_15	11	4792	1788	2268	7731
sqrt8_260	12	3009	1314	1659	5740
squar5_261	13	1993	869	1048	3644
sym6_145	7	3888	1701	2187	7615

Benchmarks	Qubits	Gates	CNOTs	DP	LC
sym9_146	12	328	148	127	450
sys6_v0_111	10	215	98	74	266
vbeAdder_2b	7	210	42	52	116
wim_266	11	986	427	514	1788
xor5_254	6	7	5	2	5
z4_268	11	3073	1343	1643	5688
adr4_197	13	3439	1498	1839	6377
9symml_195	11	34881	15232	19235	66303
clip_206	14	33827	14772	17879	61786
cm152a_212	12	1221	532	684	2366
cm85a_209	14	11414	4986	6374	21967
co14_215	15	17936	7840	8570	29608
cycle10_2_110	12	6050	2648	3384	11692
dc1_220	11	1914	833	1038	3597
dc2_222	15	9462	4131	5242	18097
dist_223	13	38046	16624	19693	68111
ham15_107	15	8763	3858	4793	16607
life_238	11	22445	9800	12511	43123
max46_240	10	27126	11844	14257	49400
mini_alu_305	10	173	77	68	242
misex1_241	15	4813	2100	2676	9240
pm1_249	14	1776	771	940	3249
radd_250	13	3213	1405	1778	6163
root_255	13	17159	7493	8835	30575
sqn_258	10	10223	4459	5458	18955
square_root_7	15	7630	3089	3830	13049
sym10_262	12	64283	28084	35572	122564
sym9_148	10	21504	9408	12087	41641

executed for five times and the one with minimum overhead is reported.

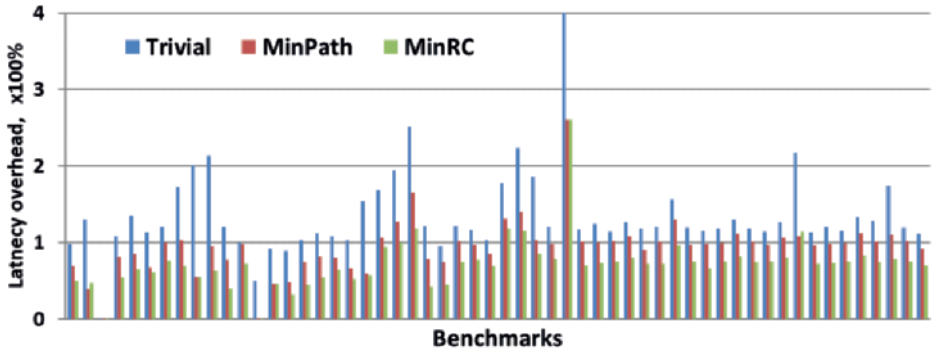
MAPPING OVERHEAD

In order to get quantum circuits which are executable on real processors, extra movement operations need to be added and gate parallelism will be compromised. We first analyze the impact of the mapping procedure in terms of number of gates, circuit latency (for Surface-17) or depth (for IBM-20) compared to the circuit characteristics before mapping in Table 7.2. As shown in Figure 7.6 and Figure 7.7, no matter which router is applied, the mapping procedure results in high overhead for most of the benchmarks. The only exceptions are the 'benstein_v' and 'graycode6_47' circuits, because some operations in these circuits can be canceled out by the optimization module in the mapper, decreasing their circuit sizes.

Mapping to Surface-17: As shown in Figure 7.6, when the Trivial router is used, the mapping leads to an overhead in the circuit latency and the total number of gates ranging from 50% ('graycode6_47') to 1160% ('xor5_254') and from 122.9% ('wim_266') to 800% ('xor5_254'), respectively. The MinPath router results in an increase of the circuit latency and the total number of gates that goes from 38.9% ('alu_v0_27') to 260% ('xor5_254')) and from 26.0% ('cuccaroAdder_1b') to 373.4% ('rd84_142')), respectively. Finally, the MinRC router increases the circuit latency and the total number of gates from 32.4% ('miller_11') to 260% ('xor5_254')) and from 20.7% ('cuccaroAdder_1b') to 78.1% ('rd32_v0')), respectively.

Mapping to IBM-20: In Figure 7.7, it is shown that the overhead in the circuit depth

(a)



(b)

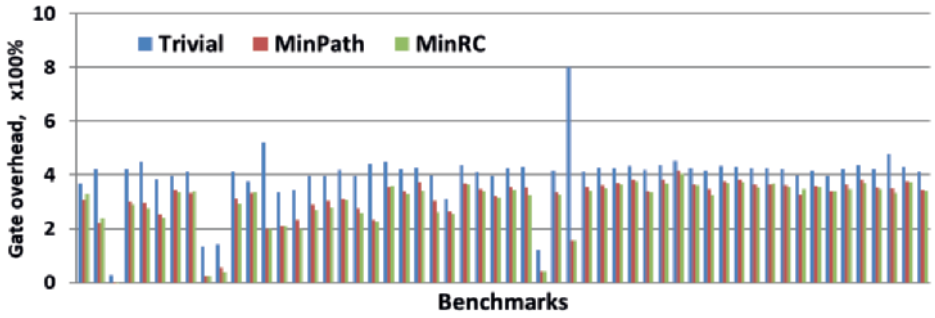


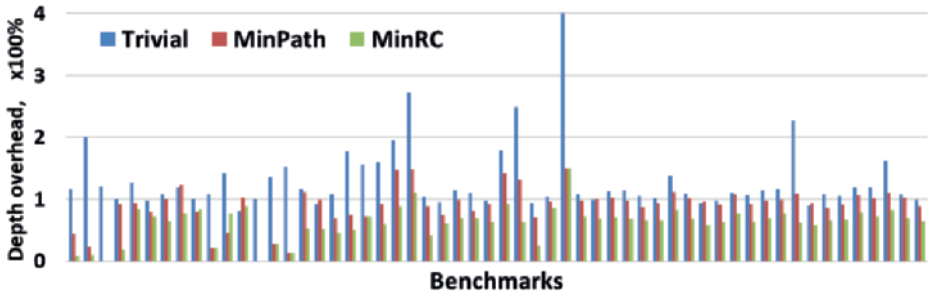
Figure 7.6: Latency and gate overhead when mapping the benchmarks in Table 7.2 on the Surface-17 processor.

and the total number of gates caused by the Trivial ranges from 80.5% ('decod24_e') to 650% ('xor5_254') and from 56.5% ('cnt3_5') to 257.1% ('xor5_254'), respectively. The circuit depth after mapping with both the based and the MinRC router has increased from 13.8% ('miller_11') to 150% ('xor5_254'). The total number of gates has increased from 10% ('ham3_102') for both routers to 72% and 68% ('rd84_142') for the MinPath router and the MinRC router, respectively.

COMPARISON OF DIFFERENT ROUTERS

We further evaluate the performance of these three different routers. As expected, for both processors, the Trivial router leads to the highest mapping overhead, as it is our baseline. It is also observed that in general the MinRC router shows the best performance as it leads to the lowest increase in circuit depth/latency and number of gates (Figure 7.6 and Figure 7.7). This is because the MinPath router includes optimizations but randomly selects one movement set. The MinRC router optimizes circuits and evaluates more shortest movement paths to select one which minimally extends the circuit

(a)



(b)

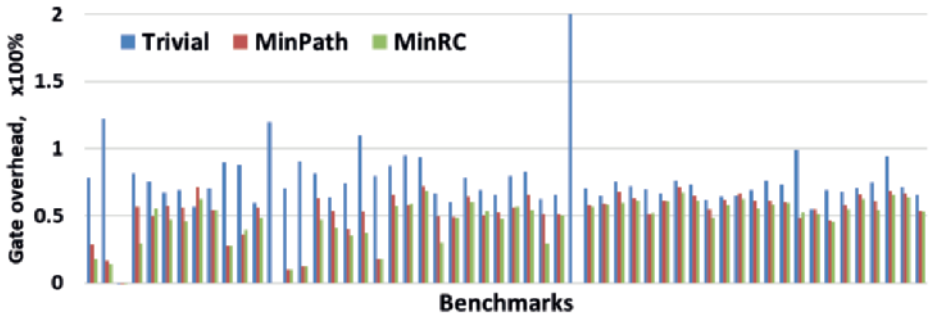


Figure 7.7: Depth and gate overhead when mapping the benchmarks in Table 7.2 on the IBM-20 processor.

latency (Section 7.3).

Mapping to Surface-17: As shown in Figure 7.6, the MinPath router always outperforms the Trivial router, the latency and the number of gates can be reduced up to 71.4% ('xor5_254') and up to 80% ('benstein_bazirani'), respectively. Moreover, the MinRC router has lower or equal overhead than the MinPath router in terms of both circuit latency and number of gates for 85.7% and 94.6% of the benchmarks, respectively. The MinRC router can reduce the latency up to 20.5% ('decod24_b') and decrease the number of gates up to 10.61% ('sf_274') compared to the MinPath router.

Mapping to IBM-20: Based on the mapping results in Figure 7.7, the MinPath router can reduce the depth for 91.1% of benchmarks (up to 66.7% for 'xor5_254') and decrease the number of gates for 94.6% of benchmarks (up to 72% for 'xor5_254') compared to the Trivial router. Furthermore, the MinRC router results in a lower or equal overhead than using the MinPath router in both circuit latency and # gates for 96.4% and 87.5% of the benchmarks, respectively. For example, the MinRC router leads to latency reduction up to 38.2% and gate reduction up to 17.4% for the benchmark '4gt12_v1_89' compared to the MinPath router.

COMPARISON OF PROCESSOR TOPOLOGY

In addition, we also investigate how the processor topology affects mapping overhead in terms of the number of inserted movement operations. For a comparison between the Surface-17 and IBM-20 processors, we transform the number of movements (SWAPs and MOVEs) into the number of elementary two-qubit gates (that is, CZ for Surface-17 and CNOT for IBM-20). Based on the mapping results shown in Table 7.4 and Table 7.5, the IBM-20 processor requires fewer movement operations than the Surface-17 processor because it has more connectivity. Figure 7.8 shows the number of inserted two-qubit gates (vertical axis) on these two processor topologies when using the MinRC router. For example, no movement operations are even needed when mapping some benchmarks ('ham3_102', 'miller_11', and 'xor5_254') to the IBM-20 processor. For other benchmarks, the IBM-20 processor can reduce the number of inserted elementary two-qubit gates up to 82.3% ('alu_v0_27') compared to the Surface-17 processor.

Note that Qmap has not been optimized for the IBM-20 processor and we leave such optimizations for future work. For example, when mapping the same benchmarks on IBM-20, Qmap has comparable performance to the SABRE mapper [146] if it does not use a more optimized initial placement. However, Qmap inserts more movement operations than the SABRE mapper when it uses this optimal initial placement.

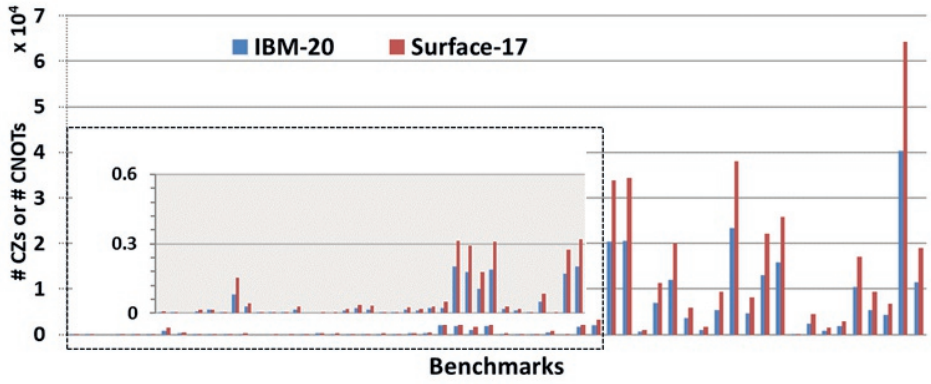


Figure 7.8: Comparison of number of inserted two-qubit gates (CZs/CNOTs) when mapping to Surface-17 and IBM-20 topologies.

RUNTIME AND SCALABILITY

We have tested the proposed mapper for different sizes of benchmarks, in which the number of qubits ranges from 3 to 16 and the two-qubit gate number from 5 to 62483. The runtime (in seconds) that Qmap requires for mapping each benchmark can be found in Table 7.4 and Table 7.5, which is measured by the CPU time that the entire mapping procedure takes (excluding the time the ILP-based initial placement takes). The router that performs more optimizations and evaluates more movement sets should have longer runtime, which is consistent with the results shown in in Table 7.4 and Table 7.5. The Trivial router has the shortest execution time, whereas the MinRC shows the longest one.

For example, for the largest benchmark 'sym10_262' with 62483 gates, the mapper using the Trivial router only takes 72.8 seconds and 5.02 seconds for the Surface-17 processor and the IBM-20 processor, respectively. In comparison, when the MinRC router is used, it takes 9083.4 seconds and 1698.4 seconds for the Surface-17 processor and the IBM-20 processor, respectively. Based on the above observation, we can conclude that our mapper is scalable in terms of large number of gates. However, our experiments only use benchmarks which have fewer than 20 qubits. Therefore, its scalability with the number of qubits needs to be further investigated. Besides, it is necessary to analyze the trade-off between mapping optimizations and runtime for large-scale benchmarks.

MOVEs VERSUS SWAPs

As mentioned in Section 7.2, a SWAP gate is implemented by three consecutive CNOT gates whereas a MOVE operation is implemented by two consecutive CNOT gates but requiring an ancilla qubit in the state $|0\rangle$. Therefore, if there are available ancilla qubits (qubits that are not used for computation), then it is preferable to use MOVE operations rather than SWAP gates, which helps to reduce the mapping overhead. In the mapping results of Tables 7.4 and 7.5, MOVE operations are allowed for both MinPath and MinRC routers. In this section, we evaluate the benefit of using MOVE operations, instead of only using SWAPs. We map the benchmarks in Table 7.2 onto the Surface-17 processor using the MinPath router. Different from the setups in Table 7.4, to have a fair comparison between using MOVEs if possible and only using SWAPs, in this case ILP-based initial placement is not applied and the first movement set is always selected. As shown in Figure 7.9 (more data can be found in Table 7.6), generating MOVEs instead of SWAPs can reduce both the number of gates up to 38.9% ('bestein_vazirani') and the circuit latency up to 29% ('graycode6_47').

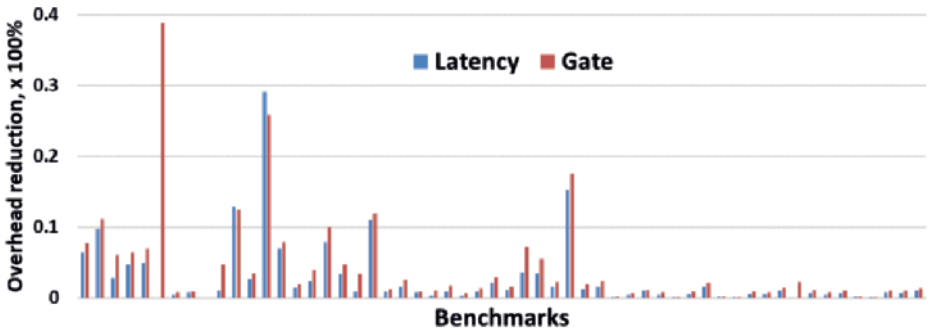


Figure 7.9: Comparison of mapping overhead on Surface-17 when using only SWAPs and using MOVEs if possible.

FIDELITY ANALYSIS

Qubits have limited coherence time and quantum operations are faulty, therefore, higher number of operations and longer circuit latency/depth will possibly lead to lower algorithm reliability which is measured by fidelity in this chapter. We investigate how the

Table 7.3: The characteristics of the benchmarks before and after mapping.

Benchmark	Before mapping				After mapping			
	Qubits	Latency	Qates	CZs	Qubits	Latency	Gates	CZs
graycode6_47	6	20	5	5	6	16	15	5
xor5_254	6	5	7	5	6	18	18	8
ham3_102	3	41	20	11	3	60	62	17
cuccroadder_1b	4	58	73	17	5	90	92	23
alu_v0_27	5	72	36	17	6	100	116	30
rd32_v0_66	4	66	34	16	6	105	113	32
miller_11	3	105	50	23	4	156	166	46

mapping affects the circuit fidelity by simulating various small benchmarks on a density-matrix-based simulator called quantumsim [24]. The error models in this simulator are implemented based on experimental parameters for transmon qubits. In this chapter, we only consider qubit decoherence (relaxation and dephasing), gate and measurement errors, using the parameters from [24]. More specifically, the qubit relaxation time T_1 and dephasing time T_ϕ are both set to be 30000 ns and 60000 ns, respectively. The in-plane error and in-axis error for single-qubit rotations are set to be $5 * 10^{-4}$ and 10^{-4}, respectively. The incoherent deviation from the expected phase value for CZ gates is $\frac{0.01}{2\pi}$ and the readout error is 0.0015.

Figure 7.10 shows the fidelity before mapping and after mapping several small-scale benchmarks (Table 7.3) on the Surface-17 processor. The fidelity is calculated by $f(\rho,\sigma) = Tr\left(\sqrt{\rho^{1/2}\sigma\rho^{1/2}}\right)$ [23], ρ and σ are the density matrix description of quantum states. As expected, the fidelity of the circuits after being mapped drops. This decrease goes from 1.8% for the 'graycode6'circuit to 13.8% for 'rd32_v0_6' and it is due to insertion of more operations and the increment of the circuit latency. Moreover, for most of the benchmarks, if a benchmark has both longer latency and more gates, then it will have lower fidelity.

These two observations suggest that circuit fidelity is correlated with the latency and the number of gates. However, other parameters may also affect the fidelity such as the number of qubits and how errors propagate through two-qubit gates, and it is not clear which one has a higher impact on it. For instance, the mapped benchmarks 'miller_11' has longer latency and more gates than 'rd32_v0_6', but it achieves higher fidelity. Another example is that the mapped benchmark 'alu_v0_27' which has shorter latency but more gates achieves higher fidelity than the mapped 'rd32_v0_6'. The impact of the mapping on the algorithm fidelity needs further investigation. The next step will be then to analyze which circuit characteristics affect (most) the fidelity, and then develop a metric which not only can well represent the fidelity but also can be easily formulated to be optimized by the mapping procedure.

7.5. RELATED WORK

As we showed, most of the works on mapping focus on IBM and Rigetti superconducting processors or on the UMD trapped ion processor. They only target a particular quantum processor (e.g. IBM Q Yorktown) or a family of processors (e.g. IBM Q Tenerife, IBM Q Melbourne, IBM Q Tokyo). Recently, mappers capable of generating executable circuits for different quantum processors have been presented [149, 154, 156]. However, none

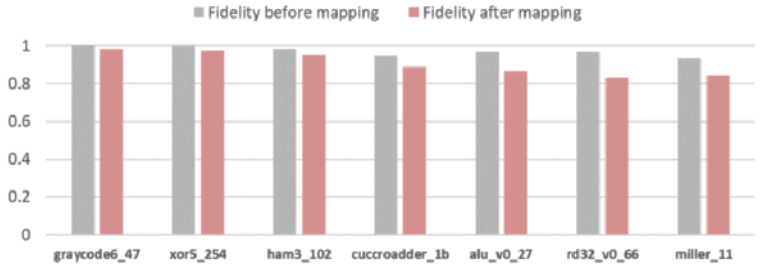


Figure 7.10: The fidelity of the benchmarks before and after mapping.

Table 7.4: The results of mapping quantum benchmarks to the Surface-17 processor, including the total number of gates and the number of two-qubit gates (CZs) in the mapped output circuits, the circuit latency (LC) in cycles (20 ns per cycle), the numbers of inserted SWAP (SWs) and MOVE (MVs) operations, and the CPU time that mapping takes in seconds.

Benchmarks	The Trivial router						The MinPath router						The MinRC router					
	LC	Gates	CZs	SWs	MVs	Time	LC	Gates	CZs	SWs	MVs	Time	LC	Gates	CZs	SWS	MVs	Time
alu_bdd_288	335	393	113	25	0	0.063651	286	341	100	16	7	1.7313	254	362	109	15	13	1.77362
alu_v0_27	166	188	56	13	0	0.035298	100	116	30	3	2	4.20968	106	122	34	3	4	4.13529
benstein_vazirani	36	45	10	3	0	0.016671	36	9	1	0	0	0.010512	36	9	1	0	0	0.011352
4gt12_v1_89	931	1191	346	82	0	0.195921	811	917	270	54	4	25.6367	690	886	259	51	3	26.0342
4gt4_v0_72	1124	1416	413	100	0	0.255498	884	1018	296	55	9	4.40794	788	973	273	52	2	4.628
4mod5_bdd_287	298	339	100	23	0	0.071202	234	247	71	10	5	18.7469	226	240	69	10	4	19.5225
cm42a_207	7167	8782	2532	587	0	1.42467	6499	7887	2352	517	15	611.534	5713	7724	2301	494	24	535.889
cnt3_5_180	1985	2491	725	170	0	0.38054	1480	2103	623	136	0	25.0301	1236	2132	641	142	0	25.6028
cuccaroAdder	175	171	50	11	0	0.030359	90	92	23	0	3	0.252098	90	92	23	0	3	0.28906
cuccaroMultiply	417	427	122	30	0	0.061224	260	274	74	10	6	2.05933	217	246	64	6	7	2.09601
decod24_bdd	315	375	110	26	0	0.063532	253	301	90	14	8	1.38109	201	287	83	15	3	1.46449
decod24_enable	1342	1607	467	106	0	0.234408	1324	1464	434	95	0	28.704	1151	1474	434	95	0	28.7617
graycode6_47	30	31	11	2	0	0.008976	16	15	5	0	0	5.83973	16	15	5	0	0	5.8601
ham3_102	79	87	26	5	0	0.011258	60	62	17	2	0	0.15724	60	62	17	2	0	0.22297
miller_11	199	222	65	14	0	0.028559	156	166	46	3	7	0.19096	139	149	39	0	8	0.27471
mini_alu_167	1144	1431	414	96	0	0.21319	985	1120	309	61	0	29.411	818	1068	294	56	0	28.5271
mod5adder_127	2229	2744	794	185	0	0.44677	1908	2240	645	130	8	7.0105	1618	2104	598	109	16	7.4544
mod8_10_177	1819	2285	661	155	0	0.368983	1570	1808	530	102	14	2.26425	1434	1786	518	106	2	2.53567
one_two_three	287	346	101	23	0	0.054458	235	263	76	12	4	6.18516	215	252	70	10	4	6.41456
rd32_v0_66	168	184	55	13	0	0.027656	105	113	32	4	2	1.65692	104	111	31	1	6	1.71454
rd53_311	1183	1514	448	108	0	0.248108	909	1249	370	78	6	0.325133	856	1257	375	81	4	0.671126
rd73_140	970	1198	350	82	0	0.190468	751	1010	300	62	5	20.682	662	988	292	52	16	20.3441
rd84_142	1385	1804	526	124	0	0.301286	1044	1624	481	109	0	20.7494	861	1516	448	98	0	21.1735
sf_274	3351	3892	1137	267	0	0.674928	2705	3157	926	178	28	40.0639	2151	2822	818	104	85	41.2879
shor_15	15082	19608	5472	1228	0	4.33023	13460	17464	5046	1028	87	2.45284	11217	17058	4924	982	95	14.6928
sqrt8_260	12708	16131	4719	1135	0	3.49803	11626	14041	4231	953	29	4.12037	10020	13944	4216	956	17	13.2009
squar5_261	7865	10178	2951	694	0	2.17597	7198	8922	2663	594	6	3.48788	6468	8764	2630	585	3	7.76352
sym6_145	15466	19266	5583	1294	0	4.12125	14094	16427	4872	965	138	3.94839	12873	16145	4787	970	88	16.7757
sym9_146	1250	1721	499	117	0	0.301726	1040	1493	447	93	10	21.1801	980	1456	431	91	5	21.6935
sys6_v0_111	859	1142	338	80	0	0.248159	640	976	290	62	3	21.1563	573	909	267	49	11	21.1608
vbeAdder_2b	332	468	135	31	0	0.097994	236	300	79	9	5	0.16455	215	298	80	6	10	0.1938
wim_266	3941	5084	1474	349	0	0.986583	3546	4289	1273	272	15	13.0377	3190	4203	1254	265	16	13.5583
xor5_254	63	63	23	6	0	0.011354	18	18	8	1	0	29.7578	18	18	8	1	0	29.2384
z4_268	12341	15792	4598	1085	0	3.19178	11463	13962	4178	905	60	818.036	9704	13537	4088	887	42	869.445
adr4_197	14296	18110	5287	1263	0	3.38715	12772	15868	4780	1082	18	1.67818	11070	15496	4685	1021	62	10.924
9symml_195	142144	182319	53224	12664	0	36.5722	134023	164219	49485	11167	376	16.642	116118	162001	49154	11282	38	2332.7
clip_206	139948	180243	52809	12679	0	40.1273	128597	162421	49227	11379	159	17.44	111253	160880	49090	11268	257	2587.99
cm152a_212	5166	6320	1834	434	0	1.3859	4508	5347	1586	346	8	0.668956	4086	5306	1591	353	0	3.53968
cm85a_209	48394	61007	17886	4300	0	14.2237	44110	54845	16654	3832	86	6.49007	37839	53363	16224	3716	45	389.036
co14_215	75821	99108	29218	7126	0	20.9755	68064	92308	28381	6837	15	10.8777	57968	90267	27787	6615	51	1044.04
cycle10_2_110	25607	31630	9236	2196	0	7.12406	23070	28148	8460	1904	50	3.26144	20458	27897	8471	1899	63	106.071
dc1_220	7740	9845	2867	678	0	2.62955	7116	8575	2574	567	20	1.45486	5979	8117	2444	481	84	8.51783
dc2_222	39466	50396	14754	3541	0	12.5991	36113	44864	13547	3100	58	5.16826	31796	44379	13520	3077	79	268.637
dist_223	156674	201426	58891	14089	0	40.4085	144079	183197	55613	12757	359	55.0312	124031	179639	54717	12599	148	3550.7
ham15_107	36221	45826	13356	3166	0	10.1604	33368	40721	12257	2797	4	5.74947	28906	39762	12030	2704	30	193.48
life_238	92286	117371	34238	8146	0	30.3595	85068	104447	31370	7134	84	14.0716	75462	104689	31920	7324	74	1364.49
max46_240	111978	141438	41211	9789	0	35.6086	101798	125209	37631	8375	331	16.8426	89164	123895	37565	8217	535	1840.89
mini_alu_305	767	862	254	59	0	0.160793	505	741	228	35	23	0.198036	518	775	242	41	21	0.409246
misex1_241	19670	24793	7206	1702	0	5.75472	18143	22002	6577	1479	20	3.13745	15892	21883	6588	1480	24	53.4152
pm1_249	7167	8782	2532	587	0	2.3615	6446	7793	2314	499	23	1.48028	5629	7774	2331	504	24	6.86838
radd_250	13254	16700	4867	1154	0	4.11447	12291	14955	4516	979	87	2.43807	10798	14408	4363	948	57	24.0061
root_255	71310	91873	26882	6463	0	20.7858	64948	82599	24991	5824	13	11.4199	55963	80542	24520	5627	73	844.114
sqn_258	43328	53252	15529	3690	0	11.7106	38165	46370	13908	3019	196	6.55198	33010	45801	13815	2984	202	270.981
square_root_7	35769	44042	12896	3269	0	10.6409	27419	34333	10274	2371	36	50077.7	23203	33088	9845	2184	102	46862.9
sym10_262	269200	340622	99658	23858	0	72.7567	247750	305153	92270	21030	548	42.2372	215185	303141	92326	20986	642	9083.41
sym9_148	87919	110393	32127	7573	0	27.4377	79881	95215	28378	6152	257	14.4444	70756	94656	28462	6182	254	800.226

Table 7.5: The results of mapping quantum benchmarks to the IBM-20 processor, including the total number of gates and the number of two-qubit gates (CNOTs) in the mapped output circuits, the circuit depth (DP), the numbers of inserted SWAP (SWs) and MOVE (MVs) operations, and the CPU time that mapping takes in seconds.

	The Trivial router						The MinPath router						The MinRC router					
Benchmarks	DP	Gates	CZs	SWs	MVs	Time	Depth	Gates	CNOTs	SWs	MVs	Time	DP	Gates	CNOTs	SWs	MVs	Time
alu_bdd_288	104	150	104	22	0	0.006799	69	108	62	6	3	1.97957	52	99	53	1	6	2.19182
alu_v0_27	63	80	59	14	0	0.003466	26	42	21	0	2	0.695886	23	41	20	1	0	0.53724
benstein_vazirani	11	19	7	2	0	0.001361	5	9	1	0	0	0.011202	5	9	1	0	0	0.006134
4gt12_v1_89	260	413	265	55	0	0.017538	251	357	225	39	4	1.01012	155	295	163	17	6	1.08303
4gt4_v0_72	310	453	302	63	0	0.018983	265	386	239	42	0	1.31254	251	401	254	47	0	1.43037
4mod5_bdd_287	79	117	70	13	0	0.006572	72	110	69	8	7	10.5762	69	103	62	7	5	11.2424
cm42a_207	1951	2997	1914	381	0	0.132164	1885	2771	1752	325	3	1.54932	1552	2591	1572	267	0	2.04996
cnt3_5_180	453	759	449	78	0	0.031661	461	832	526	95	13	0.251479	366	787	481	84	7	0.403158
cuccaroAdder_1b	50	63	35	6	0	0.002176	45	57	29	4	0	0.006341	46	57	29	4	0	0.006140
cuccaroMultiply	114	163	95	21	0	0.008113	67	110	42	2	2	0.063068	67	110	42	2	2	0.064569
decod24_bdd_294	97	137	92	20	0	0.005843	58	99	54	4	5	0.220697	71	102	57	7	2	0.265615
decod24_enable	343	539	320	57	0	0.019654	385	528	325	52	10	0.282428	359	501	298	41	13	0.312647
graycode6_47	10	11	11	2	0	0.001782	5	5	5	0	0	132.424	5	5	5	0	0	134.398
ham3_102	26	34	23	4	0	0.001695	14	22	11	0	0	0.272413	14	22	11	0	0	0.335248
miller_11	73	95	62	13	0	0.003827	33	56	23	0	0	0.398531	33	56	23	0	0	0.394078
mini_alu_167	352	523	333	69	0	0.021998	343	469	287	47	10	0.358596	248	423	241	23	23	0.287731
mod5adder_127	581	909	557	106	0	0.038077	600	853	517	82	16	0.347671	460	783	447	52	26	0.429595
mod8_10_177	515	768	490	98	0	0.030661	422	616	350	44	11	1.38649	362	595	329	27	26	1.39496
one_two_three	111	147	101	23	0	0.006499	70	107	61	7	4	15.6825	60	96	50	4	3	15.3512
rd32_v0_66	46	61	43	9	0	0.003419	31	40	22	2	0	0.335373	31	40	22	2	0	0.261868
rd53_311	323	515	346	74	0	0.021726	238	455	286	50	6	0.226095	198	433	264	42	7	0.354412
rd73_140	272	449	299	65	0	0.017987	227	363	225	39	2	0.215851	174	365	227	41	0	0.248996
rd84_142	409	665	442	96	0	0.032003	274	590	375	73	1	0.226931	232	578	363	69	1	0.291646
sf_274	891	1303	810	158	0	0.050961	821	1168	715	121	8	0.292848	615	1014	561	55	30	0.505735
shor_15	4412	7660	4110	774	0	0.298939	3962	7143	3821	675	4	0.538582	3642	7121	3799	657	20	3.94586
sqrt8_260	3557	5362	3507	731	0	0.223606	3295	4941	3218	632	4	0.46522	2806	4823	3100	588	11	2.80964
squar5_261	2204	3369	2159	430	0	0.14577	1902	2996	1868	333	0	0.146514	1772	3056	1928	353	0	1.25905
sym6_145	4308	6442	4089	796	0	0.270245	4214	5931	3746	639	64	0.566357	3578	5746	3561	580	60	3.62793
sym9_146	355	589	385	79	0	0.025852	308	512	312	54	1	0.225797	244	513	313	51	6	0.301447
sys6_v0_111	258	392	257	53	0	0.016768	172	356	221	39	3	0.22927	121	332	197	27	9	0.366101
vbeAdder_2b	101	166	90	16	0	0.009988	89	154	82	6	11	0.08707	65	132	60	4	3	0.097052
wim_266	1050	1634	1027	200	0	0.068294	1006	1490	927	160	10	0.31237	959	1477	914	159	5	0.601171
xor5_254	15	25	23	6	0	0.001625	5	7	5	0	0	83.2281	5	7	5	0	0	86.1867
z4_268	3416	5233	3353	670	0	0.211396	3242	4844	3096	573	17	1.15126	2830	4807	3059	562	15	3.34761
adr4_197	3663	5663	3556	686	0	0.289353	3690	5471	3547	683	0	0.70171	3107	5447	3523	675	0	3.79805
9symml_195	40942	61211	39568	8112	0	2.41685	38966	58625	38575	7733	72	3.29246	32774	55634	35584	6778	9	453.364
clip_206	38219	58228	37401	7543	0	2.41549	35271	55144	35837	6973	73	4.69567	30193	54562	35255	6759	103	516.907
cm152a_212	1405	2071	1330	266	0	0.096682	1280	1849	1156	200	12	0.39819	1132	1852	1159	203	9	1.02137
cm85a_209	12801	19022	12036	2350	0	0.870193	12377	18424	11982	2258	111	1.41441	10576	18321	11879	2251	70	49.9608
co14_215	20373	31624	20488	4216	0	1.46214	18173	30707	20503	4199	33	2.11778	15606	30000	19796	3968	26	145.909
cycle10_2_110	7092	10474	6704	1352	0	0.483386	6794	9961	6447	1265	2	0.98825	5701	9757	6243	1197	2	11.4825
dc1_220	2007	3096	1919	362	0	0.151444	2032	2964	1883	344	9	0.40237	1647	2834	1753	300	10	1.6554
dc2_222	10331	15520	9687	1852	0	0.868323	10023	15292	9879	1880	54	0.99071	8527	14914	9501	1748	63	30.287
dist_223	41330	62812	39502	7626	0	2.55385	40875	63362	41536	8302	3	603.305	34997	61730	39904	7758	3	980.527
ham15_107	9919	14832	9453	1865	0	0.91102	9198	14159	9128	1740	25	1.15454	7810	13622	8591	1569	13	18.3199
life_238	26825	39551	25736	5312	0	1.94568	24642	36245	23410	4522	22	1.97507	21195	35599	22764	4310	17	132.28
max46_240	30804	47036	30210	6122	0	2.29855	28361	43463	27789	5311	6	2.19886	25289	43285	27611	5253	4	175.946
mini_alu_305	223	344	242	55	0	0.018605	142	257	157	22	7	0.26750	110	264	164	27	3	0.35919
misex1_241	5080	7440	4563	821	0	0.504028	5171	7465	4736	862	25	0.36909	4232	7271	4542	800	21	5.42279
pm1_249	1951	2997	1914	381	0	0.218987	1744	2604	1585	270	2	1.3032	1556	2586	1567	252	20	2.43371
radd_250	3651	5395	3451	682	0	0.361469	3402	5075	3279	624	1	0.59758	2975	4979	3183	592	1	3.43631
root_255	19367	29259	18779	3762	0	1.51796	18265	28528	18692	3733	0	1.87795	15814	27832	17996	3501	0	81.9624
sqn_258	11936	17901	11569	2370	0	0.836415	10989	16400	10486	2005	6	1.26047	9368	15742	9828	1781	13	26.8857
square_root_7	10066	14833	9320	2077	0	0.690437	8067	12862	7540	1463	31	600.994	6995	12629	7307	1382	36	618.468
sym10_262	74050	110071	70402	14106	0	5.02098	71921	106926	70006	13968	9	6.19955	60532	105318	68398	13434	6	1698.4
sym9_148	24017	35547	22683	4425	0	1.83779	22827	33047	20947	3841	8	2.19047	19952	32947	20847	3799	21	170.922

of them take into account the control electronics constraints that can be very restrictive especially when scaling-up quantum processors. Neither do they consider information such as gate duration (except [156]) and they assume when scheduling operations that all gates take the same number of cycles to execute. They all use SWAP operations for moving qubits when targeting superconducting quantum processors. In addition, so far no mapper has been developed for more scalable quantum processors such as the Surface-17 processor presented in [39, 157]. This processor has been designed with the aim of building a large qubit array capable of performing fault-tolerant quantum computations based on surface code. It can be used not only for performing QEC cycles (memory) but also for running quantum algorithms.

Moreover, many existing mapping algorithms [72, 73, 74, 75, 76, 77, 78, 79, 80, 82, 83, 130, 144, 81, 145, 146, 147, 148, 149] usually use either the number of inserted movement

Table 7.6: Comparison of mapping results with and without using MOVE operations, including the total number of gates and the number of two-qubit gates (CZ) in the mapped output circuits, the circuit latency in cycles (20 ns per cycle), and the numbers of inserted SWAP and MOVE operations.

Benchmarks	MOVE operations are not used					MOVE operations are used				
	Latency	Gates	CZs	SWAPs	MOVEs	Latency	Gates	CZs	SWAPs	MOVEs
alu_bdd_288	340	375	110	24	0	324	351	104	18	6
alu_v0_27	144	159	47	10	0	137	148	44	7	3
4gt12_v1_89	822	992	286	62	0	770	916	265	41	21
4gt4_v0_72	1021	1237	362	83	0	921	1100	325	46	37
4mod5_bdd_287	289	315	94	21	0	281	296	89	16	5
benstein_vazirani	36	36	10	3	0	36	22	7	0	3
cm42a_207	6413	7778	2256	495	0	6387	7723	2242	481	14
cnt3_5_180	1523	2176	629	138	0	1512	2157	625	134	4
cuccaroAdder_1b	145	139	41	8	0	145	139	41	8	0
cuccaroMultiply	398	408	122	30	0	394	389	118	26	4
decod24_bdd	287	338	98	22	0	250	296	87	11	11
decod24_enable	1266	1442	413	88	0	1233	1393	400	75	13
graycode6_47	31	31	11	2	0	22	23	9	0	2
ham3_102	87	90	26	5	0	81	83	24	3	2
miller_11	206	212	62	13	0	203	208	61	12	1
mini_alu_167	1043	1216	348	74	0	1019	1168	336	62	12
mod5adder_127	2187	2541	737	166	0	2017	2287	670	99	67
mod8_10_177	1614	1968	568	124	0	1560	1875	544	100	24
one_two_three	230	295	83	17	0	228	285	80	14	3
rd32_v0_66	174	186	55	13	0	155	164	50	8	5
rd53_311	1064	1369	409	95	0	1055	1353	406	92	3
rd73_140	827	1114	326	74	0	814	1086	319	67	7
rd84_142	1115	1752	532	126	0	1107	1736	527	121	5
sf_274	2923	3366	975	213	0	2914	3333	967	205	8
shor_15	13498	18208	5292	1168	0	13377	17901	5214	1090	78
sqrt8_260	11292	13616	3984	890	0	11255	13532	3962	868	22
squar5_261	7081	8798	2573	568	0	7022	8685	2546	541	27
sym6_145	14751	17038	4944	1081	0	14447	16552	4808	945	136
sym9_146	1085	1557	460	104	0	1073	1533	454	98	6
sys6_v0_111	664	1052	314	72	0	640	977	294	52	20
vbeAdder_2b	344	475	144	34	0	332	449	138	28	6
wim_266	3605	4377	1276	283	0	3548	4282	1250	257	26
xor5_254	59	63	23	6	0	50	52	20	3	3
z4_268	11199	13812	4049	902	0	11069	13554	3979	832	70
adr4_197	12599	16008	4702	1068	0	12401	15640	4603	969	99
9symml_195	130111	156596	45649	10139	0	129997	156383	45587	10077	62
clip_206	126884	156543	46107	10445	0	126315	155589	45843	10181	264
cm152a_212	4406	5364	1555	341	0	4361	5303	1538	324	17
cm85a_209	42879	51691	15120	3378	0	42681	51300	15008	3266	112
co14_215	66055	87121	26026	6062	0	66047	87102	26022	6058	4
cycle10_2_110	22993	27807	8144	1832	0	22873	27573	8082	1770	62
dc1_220	7147	8665	2534	567	0	7040	8484	2484	517	50
dc2_222	35799	43119	12660	2843	0	35746	43042	12640	2823	20
dist_223	139285	174072	51133	11503	0	139228	173943	51100	11470	33
ham15_107	32885	39831	11658	2600	0	32699	39486	11562	2504	96
life_238	83192	100627	29378	6526	0	82711	99879	29170	6318	208
max46_240	99680	122065	35649	7935	0	98653	120295	35145	7431	504
mini_alu_305	537	804	236	53	0	537	786	232	49	4
misex1_241	17982	21417	6234	1378	0	17871	21191	6172	1316	62
pm1_249	6413	7778	2256	495	0	6387	7723	2242	481	14
radd_250	12103	14626	4285	960	0	12030	14488	4247	922	38
root_255	63744	79324	23414	5307	0	63640	79130	23361	5254	53
sqn_258	37976	45258	13153	2898	0	37964	45244	13150	2895	3
square_root_7	31238	37388	11393	2768	0	31002	37017	11305	2680	88
sym10_262	241971	293917	86188	19368	0	240301	290933	85358	18538	830
sym9_148	79993	95059	27477	6023	0	79257	93815	27137	5683	340

operations or the circuit depth as optimization metric; that is, the routing path that inserts the least number of extra gates or the one that produces the minimal circuit depth

overhead is chosen. The same metrics together with the execution time (time it takes to perform the mapping) are considered to evaluate the quality of the mapping algorithms. Although both number of gates and circuit depth are correlated with the reliability of quantum circuits and they should be minimized as we mentioned, an analysis on how they degrade the algorithm's performance is not provided.

Recent works [71, 145, 147, 148, 155, 154], propose to use reliability as an optimization metric and analyze how the mapping process affects the success rate (also called execution success probability) of the algorithm. They suggest to choose the routing path based on the fidelity of the two-qubit gates along the path as they are used to implement the movements (noise-aware mapper). Note that the fidelity of two-qubit gates can vary between different pairs of qubits. However, the reliability of a path is calculated by simply multiplying the reliability of each gate without considering error propagation and decoherence, which makes this metric incomplete and not very accurate; it sometimes fails in predicting the most reliable route [148]. Based on the results presented in these papers, it seems that optimizing reliability instead of just number of gates leads to better success rates, at least for small quantum circuits.

7.6. CONCLUSION AND DISCUSSION

In this chapter, we have presented a mapper called Qmap to make quantum circuits executable on the Surface-17 chip. It takes into account common processor constraints such as the elementary gate set and qubit connectivity, as well as classical control electronics restrictions. Qmap has been embedded in the OpenQL compiler and consists of several modules, including qubit initial placement and routing, operation scheduling, and gate decomposition and optimization. It can be applied to different processors of which hardware constraints are described in a configuration file.

We mapped 56 quantum benchmarks on two superconducting processors, which are the surface-17 processor and the IBM Q Tokyo processor. Three different routers, namely, Trivial, MinPath, and MinRC, were used in this evaluation by the Qmap mapper. For both processors, the mapping using the MinRC router results in the lowest overhead in terms of both circuit latency/depth and number of gates. Furthermore, as expected, the IBM-20 processor requires fewer movement operations compared to the Surface-17 processor due to its slightly higher qubit connectivity. We also showed that the use of a cheaper movement operation (MOVE) helps to substantially reduce the resulting overhead in terms of both added gates and latency.

We can then conclude that a flexible mapper is required for making quantum circuits executable on different real quantum processors. It must consider not only processor restrictions but also control electronic constraints as they may limit the parallelism of the operations. In addition, evaluating all possible shortest paths and different movement sets within each path and choosing one based on how well it interleaves with previous operations (look-back feature), lead to lower number of gates and circuit latency/depth. As shown, these two metrics seem to be correlated with the reliability of the algorithm, but a deeper analysis is required to develop an accurate reliability metric that can be directly used by the mapping procedure. Finally, optimizations for reducing the number of operations at different steps of the mapping process are also necessary.

Although our mapper has shown the capability to map benchmarks with large num-

ber of operations, we need to make it scalable for larger number of qubits. Future work will also include the improvement of the initial placement and routing by, for instance, finding movement operations for several two-qubit gates simultaneously. Furthermore, additional mapping metrics need be investigated and included in the mapper. Note that what parameter(s) to optimize during the mapping might depend on the characteristics of the target quantum processor. In addition, our mapping approach is based on the compilation of quantum circuits at the gate level, where the generated instructions are further translated by the microarchitecture into appropriate signals that control the qubits [106]. A different approach is to directly compile quantum algorithms to control pulses [123]. Further work will compare both solutions and investigate the trade-off of allocating mapping tasks to a compiler and a microarchitecture.

8

CONCLUSION AND OUTLOOK

THis thesis has investigated fault-tolerant protocols for performing reliable quantum computation and proposed mapping techniques for executing quantum algorithms on real processors. This chapter first summarizes the main contributions of this thesis and then discusses some future research topics.

8.1. CONCLUSION

The first part of this thesis has focused on the execution of large-scale quantum algorithms which require quantum error correction and fault-tolerant mechanisms. The main results of this part are summarized as follows:

- We started by introducing fault-tolerant quantum computation based on rotated planar surface codes in Chapter 3. The implementation of a universal set of logical operations was illustrated. These operations include the CNOTgate realized by an approach called lattice surgery, the H gate using a presented lattice rotation procedure based on code deformation, and the S and T gates which require magic state preparation. Furthermore, we proposed to describe lattice-surgery-based and code-deformation-based operations using the gauge fixing technique, showing that the fault tolerance of these protocols is determined by the underlying subsystem codes. In addition, we numerically demonstrated our approach and evaluated these logical operations under phenomenological noise and circuit level noise using the stabilizer formalism.

- The surface code has high error threshold for both logical memory and logical operations. Moreover, the error syndrome measurement for surface codes only requires local connectivity of physical qubits, which can be directly supported by many quantum technologies. However, the logical CNOTgates using lattice surgery needs logical qubits to be placed in particular neighboring positions. This connectivity restriction at the logical level requires the movement of logical qubits, which causes overhead in terms of both the number of operations and circuit latency and in turn increases logical error rates. In Chapter 4, we proposed two qubit plane architectures that organize qubits in specialized areas to efficiently support these lattice-surgery-based operations, namely, checkerboard architecture and a tile-based architecture. We further developed a full mapping procedure, including placement and routing of logical qubits and scheduling of logical operations, to map logical quantum circuits onto these two architectures. The evaluation results have shown that the tile-based architecture that uses more qubits leads to lower mapping overhead compared to the checkerboard one. More specifically, it can reduce the operation overhead and the latency overhead up to ~ 86% and ~ 79%, respectively.

- We proposed a microarchitecture, FT-QuMA, to enable the execution of fault-tolerant error correction and computation on rotated planar surface codes. It can support flexible logical operation description, efficient error syndrome extraction circuit generation, error decoding, and error correction using Pauli frame. In this control microarchitecture, the concept of virtual memory was adopted to provide a clean compilation model that is independent of actual physical qubit addresses.

Moreover, QEC increases the number of operations on qubits per unit time, aggravating the instruction issue rate problem. We presented a two-level address mechanism which can substantially reduce the codesize of the executable to enable efficient execution of quantum instructions. Finally, we verified the proposed control microarchitecture by illustrating an instantiation of the lattice-surgery-based CNOTgate.

The surface code is promising for large-scale quantum computation but it requires many physical qubits to encode one logical qubit. Noisy intermediate-scale quantum processors will only have up to a few hundreds of qubits and the connectivity between them will also be limited. Therefore, a fault-tolerant quantum computer that can run large-scale quantum algorithms using surface codes is still far away. The second part of this thesis has discussed quantum computation on NISQ processors, the main contributions are the following:

- We first investigated small-size QEC codes that can be used to demonstrate fault tolerance in small quantum processors. One possible solution is to apply the flag-based protocol which can perform quantum error correction fault-tolerantly by only using a small number of qubits. However, these flag circuits may not be directly executable on the connectivity-constrained processors. General mapping algorithms that insert SWAPoperations will likely destroy the fault tolerance of these circuits. In Chapter 6, we proposed a flag-bridge scheme that can fault-tolerantly cope with this connectivity constraint with no or low qubit overhead. Based on this proposed approach, we mapped the Steane code error correction to the Surface-17 transmon processor and the IBM Q Tokyo processor. Furthermore, we numerically simulated these mapped circuits under circuit level noise. In this simulation, a look-up-table decoder and a neural network decoder were designed. The numerical results have shown that the flag-bridge circuits that can measure more stabilizers in parallel have lower logical error rates.

- Subsequently, we focused on the implementation of quantum algorithms without error correction (Chapter 7). Besides the noisy property and limited connectivity, NISQ processors also have other hardware constraints such as predefined elementary gates and shared classical electronic control. Quantum circuits need to be compiled to respect all of these constraints, which is the mapping problem on the physical level. We developed a mapper for adapting quantum circuits to superconducting processors and three routing strategies were designed. It uses a configuration file to describe all the hardware constraints such that it is applicable to different platforms. We evaluated this mapper by mapping more than fifty benchmarks onto both the Surface-17 transmon processor and the IBM Q Tokyo processor. The mapping results has shown that the proposed mapper can reduce operation overhead up to 80% (72%) and the latency (depth) overhead up to 71.4% (66.7%) on the Surface-17 (IBM Q Tokyo), respectively, compared to a baseline mapper.

8.2. OUTLOOK

In this thesis, we have proposed protocols for performing fault-tolerant computation on large-scale and near-term quantum processors. We have also developed mapping passes and microarchitectural blocks to make quantum circuits executable on real quantum processors. Although these schemes and approaches have been well investigated and thoroughly verified, there are still many open questions and interesting research topics. Some research recommendations for future work are:

- As discussed in Chapters 3 and 6, the numerical analysis of quantum error correction and quantum computation in this thesis was carried out with Pauli error models that inject errors in a probabilistic way. The noise in real quantum devices is however continuous. Previous works have simulated the memory performance of small QEC codes with realistic error models including amplitude damping and coherent noise [158, 159, 160, 161]. The next steps are to analyze the fault tolerance of the logical operations on rotated surface codes using realistic errors. Furthermore, it is also interesting to test whether the proposed flag-bridge error correction protocol can still be fault-tolerant with realistic error models and even to perform experiments on real devices. In addition, the flag-bridge approach has only been verified on several distance-3 codes, its extensibility and scalability to higher code distances and logical operations need to be investigated.

- We have proposed qubit plane architectures and mapping passes to support fault-tolerant quantum computation by using the lattice surgery technique on rotated planar surface codes. In the mapping procedure, we assume that the magic states have been prepared whenever S and T gates will be performed. Future work needs to consider the dynamics of magic state preparation and investigate its implications on both quantum circuit mapping and quantum control microarchitecture. Another research question is where to allocate different tasks, on the quantum compiler or the control microarchitecture. For example, the translation of logical operations into physical ones as well as the mapping of quantum circuits can be performed by the compiler as the design proposed in this thesis. These tasks can also be taken by the microarchitecture, which leaves a simple interface to the quantum software but imposes challenges to the microarchitecture.

- We have developed a mapper to efficiently map quantum circuits onto NISQ processors (e.g. the Surface-17 superconducting processor) that have many hardware constraints including elementary gate set, qubit connectivity, and shared classical control. However, there is a lot of room for further optimizing the compilation results. For instance, one can improve routing strategies by finding movement operations for multiple two-qubit gates at the same time rather than one gate per time. Moreover, existing mapping algorithms (including the ones in this thesis) are mostly using the number of operations or the circuit depth/latency as optimization objectives. These metrics are all related to the reliability (fidelity and success rate) of the given quantum circuit, but which ones affect reliability most has not been studied. Another research direction is to investigate the correlation between

reliability and different circuit parameters and then define a metric which can not only better represent reliability but also be easily used by the mapping procedure.

- There are also different techniques for solving the research questions of this thesis. Regarding the execution of surface-code-based quantum circuits, one can use circuit synthesis algorithms to avoid the moving of qubits as presented in [162]. This approach translates all the gates into T gates with local multi-qubit measurement (implemented by lattice surgery) such that routing of qubits is not required. It will be interesting to compare it with our solution in terms of spatial and temporal cost as well as reliability. Additionally, there are many other quantum error correction codes that have good properties. For instance, though the color codes currently have lower threshold than surface codes, they allow transversal Clifford gates in two dimensions [163, 164] and transversal non-Clifford gates in three dimensions [165, 166]. The quantum low-density parity check (LDPC) codes have constant rate and efficient decoding algorithms [167, 168, 169]. It will be worth to investigate how to perform fault-tolerant computation on quantum LDPC codes.

REFERENCES

[1] Peter W Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 124–134. IEEE, 1994.

[2] Stephen Jordan. Quantum algorithm zoo. `http://math.nist.gov/quantum/zoo/`, 2011.

[3] Julian Kelly, Rami Barends, Austin G Fowler, Anthony Megrant, Evan Jeffrey, Theodore C White, Daniel Sank, Josh Y Mutus, Brooks Campbell, Yu Chen, et al. State preservation by repetitive error detection in a superconducting quantum circuit. *Nature*, 519(7541):66, 2015.

[4] Diego Ristè, Stefano Poletto, M-Z Huang, Alessandro Bruno, Visa Vesterinen, O-P Saira, and Leonardo DiCarlo. Detecting bit-flip errors in a logical qubit using stabilizer measurements. *Nature communications*, 6:6983, 2015.

[5] Abhinav Kandala, Antonio Mezzacapo, Kristan Temme, Maika Takita, Jerry M Chow, and Jay M Gambetta. Hardware-efficient quantum optimizer for small molecules and quantum magnets. *arXiv:1704.05018*, 2017.

[6] Ronald Hanson, Leo P Kouwenhoven, Jason R Petta, Seigo Tarucha, and Lieven MK Vandersypen. Spins in few-electron quantum dots. *Reviews of modern physics*, 79(4):1217, 2007.

[7] Floris A Zwanenburg, Andrew S Dzurak, Andrea Morello, Michelle Y Simmons, Lloyd CL Hollenberg, Gerhard Klimeck, Sven Rogge, Susan N Coppersmith, and Mark A Eriksson. Silicon quantum electronics. *Reviews of Modern Physics*, 85:961, 2013.

[8] TF Watson, SGJ Philips, Erika Kawakami, DR Ward, Pasquale Scarlino, Menno Veldhorst, DE Savage, MG Lagally, Mark Friesen, SN Coppersmith, et al. A programmable two-qubit quantum processor in silicon. *Nature*, 555(7698):633, 2018.

[9] G De Lange, ZH Wang, D Riste, VV Dobrovitski, and R Hanson. Universal dynamical decoupling of a single solid-state spin from a spin bath. *Science*, 330:60–63, 2010.

[10] Julia Cramer, Norbert Kalb, Michiel Adriaan Rol, Bas Hensen, Machiel S Blok, Matthew Markham, Daniel J Twitchen, Ronald Hanson, and Tim H Taminiau. Repeated quantum error correction on a continuously encoded qubit by real-time feedback. *Nature Communications*, 7, 2016.

[11] Chris Monroe, DM Meekhof, BE King, Wayne M Itano, and David J Wineland. Demonstration of a fundamental quantum logic gate. *Phys. Rev. Lett.*, 75:4714, 1995.

[12] Shantanu Debnath, Norbert M Linke, Caroline Figgatt, Kevin A Landsman, Kevin Wright, and Christopher Monroe. Demonstration of a small programmable quantum computer with atomic qubits. *Nature*, 536(7614):63, 2016.

[13] Antonio D Córcoles, Easwar Magesan, Srikanth J Srinivasan, Andrew W Cross, Matthias Steffen, Jay M Gambetta, and Jerry M Chow. Demonstration of a quantum error detection code using a square lattice of four superconducting qubits. *Nature communications*, 6:6979, 2015.

[14] IBM. Quantum experience. `https://www.research.ibm.com/ibm-q/`, 2017.

[15] Austin G Fowler, Matteo Mariantoni, John M Martinis, and Andrew N Cleland. Surface codes: Towards practical large-scale quantum computation. *Phys. Rev. A*, 86(3):032324, 2012.

[16] Rami Barends, Julian Kelly, Anthony Megrant, Andrzej Veitia, Daniel Sank, Evan Jeffrey, Ted C White, Josh Mutus, Austin G Fowler, Brooks Campbell, et al. Superconducting quantum circuits at the surface code threshold for fault tolerance. *Nature*, 508(7497):500–503, 2014.

[17] Charles D Hill, Eldad Peretz, Samuel J Hile, Matthew G House, Martin Fuechsle, Sven Rogge, Michelle Y Simmons, and Lloyd CL Hollenberg. A surface code quantum computer in silicon. *Science advances*, 1(9):e1500707, 2015.

[18] Clare Horsman, Austin G Fowler, Simon Devitt, and Rodney Van Meter. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14(12):123011, 2012.

[19] Andrew J Landahl and Ciaran Ryan-Anderson. Quantum computing by color-code lattice surgery. *arXiv:1407.5103*, 2014.

[20] Héctor Bombín and Miguel A Martin-Delgado. Quantum measurements and gates by code deformation. *Journal of Physics A: Mathematical and Theoretical*, 42(9):095302, 2009.

[21] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.

[22] Xiang Fu, Leon Riesebos, Lingling Lao, Carmen G Almudever, Fabio Sebastiano, Richard Versluis, Edoardo Charbon, and Koen Bertels. A heterogeneous quantum computer architecture. In *Proceedings of the ACM International Conference on Computing Frontiers*, pages 323–330. ACM, 2016.

[23] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge University Press, 2010.

[24] Thomas E O'Brien, Brian Tarasinski, and Leonardo DiCarlo. Density-matrix simulation of small surface codes under current and projected experimental noise. *npj Quantum Information*, 3(1):39, 2017.

[25] Daniel Gottesman. The heisenberg representation of quantum computers. *arXiv:9807006*, 1998.

[26] Andrew M Steane. Error correcting codes in quantum theory. *Phys. Rev. Lett.*, 77(5):793, 1996.

[27] Dave Bacon. Operator quantum error-correcting subsystems for self-correcting quantum memories. *Phys. Rev. A*, 73(1):012340, 2006.

[28] Alexei Kitaev. Fault-tolerant quantum computation by anyons. *Annals of Physics*, 303(1):2–30, 2003.

[29] Daniel Gottesman. Stabilizer codes and quantum error correction. *arXiv:9705052*, 1997.

[30] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. Topological quantum memory. *Journal of Mathematical Physics*, 43(9):4452–4505, 2002.

[31] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17(3):449–467, 1965.

[32] Francisco Barahona, R Maynard, R Rammal, and JP Uhry. Morphology of ground states of two-dimensional frustration model. *Journal of Physics A: Mathematical and General*, 15(2):673, 1982.

[33] Vladimir Kolmogorov. Blossom V: a new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 1:43–67, 2009.

[34] Guillaume Duclos-Cianci and David Poulin. A renormalization group decoding algorithm for topological quantum codes. In *Information Theory Workshop (ITW), 2010 IEEE*, pages 1–5. IEEE, 2010.

[35] Sergey Bravyi, Martin Suchara, and Alexander Vargo. Efficient algorithms for maximum likelihood decoding in the surface code. *Phys. Rev. A*, 90(3):032326, 2014.

[36] Savvas Varsamopoulos, Ben Criger, and Koen Bertels. Decoding small surface codes with feedforward neural networks. *Quantum Science and Technology*, 3(1):015004, 2017.

[37] Xiaotong Ni. Neural network decoders for large-distance 2d toric codes. *arXiv:1809.06640*, 2018.

[38] Emanuel Knill. Quantum computing with very noisy devices. *arXiv:0410199*, 2004.

[39] Richard Versluis, Stefano Poletto, Nader Khammassi, Brian Tarasinski, Nadia Haider, David J Michalak, Alessandro Bruno, Koen Bertels, and Leonardo DiCarlo. Scalable quantum circuit and control for a superconducting surface code. *Phys. Rev. Applied*, 8(3):034021, 2017.

[40] Sergey Bravyi and Alexei Kitaev. Universal quantum computation with ideal Clifford gates and noisy ancillas. *Phys. Rev. A*, 71(2):022316, 2005.

[41] Sergey Bravyi and Jeongwan Haah. Magic-state distillation with low overhead. *Phys. Rev. A*, 86(5):052329, 2012.

[42] Adam M Meier, Bryan Eastin, and Emanuel Knill. Magic-state distillation with the four-qubit code. *arXiv:1204.4221*, 2012.

[43] Cody Jones. Multilevel distillation of magic states for quantum computing. *Phys. Rev. A*, 87(4):042305, 2013.

[44] Earl T Campbell and Mark Howard. Unifying gate synthesis and magic state distillation. *Phys. Rev. Lett.*, 118(6):060501, 2017.

[45] Eyob A Sete, William J Zeng, and Chad T Rigetti. A functional architecture for scalable quantum computing. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–6. IEEE, 2016.

[46] Ruoyu Li, Luca Petit, David P Franke, Juan Pablo Dehollain, Jonas Helsen, Mark Steudtner, Nicole K Thomas, Zachary R Yoscovits, Kanwal J Singh, Stephanie Wehner, et al. A crossbar network for silicon quantum dot qubits. *Science advances*, 4(7):eaar3960, 2018.

[47] Christopher Monroe and Jungsang Kim. Scaling the ion trap quantum processor. *Science*, 339(6124):1164–1169, 2013.

[48] Krysta M Svore, Alfred V Aho, Andrew W Cross, Isaac Chuang, and Igor L Markov. A layered software architecture for quantum computing design tools. *Computer*, pages 74–83, 2006.

[49] Emanuel Knill and Raymond Laflamme. Concatenated quantum codes. *arXiv:9608012*, 1996.

[50] David S Wang, Austin G Fowler, and Lloyd CL Hollenberg. Surface code quantum computing with error rates over 1%. *Phys. Rev. A*, 83:020302, 2011.

[51] Adam Paetznick and Ben W Reichardt. Universal fault-tolerant quantum computation with only transversal gates and error correction. *Phys. Rev. Lett.*, 111(9):090505, 2013.

[52] Héctor Bombín. Clifford gates by code deformation. *New Journal of Physics*, 13(4):043005, 2011.

[53] Sergey Bravyi. Fault-tolerant quantum computing by code deformation. *QIP Tutorial*, 2016.

[54] Hendrik Poulsen Nautrup, Nicolai Friis, and Hans J. Briegel. Fault-tolerant interface between quantum memories and quantum processors. *Nature Communications*, 8(1):1321, 2017.

[55] Benjamin J Brown, Katharina Laubscher, Markus S Kesselring, and James R Wootton. Poking holes and cutting corners to achieve Clifford gates with the surface code. *Phys. Rev. X*, 7:021029, May 2017.

[56] Austin G Fowler and Craig Gidney. Low overhead quantum computation using lattice surgery. *arXiv:1808.06709*, 2018.

[57] Michael Vasmer and Dan E Browne. Universal quantum computing with 3d surface codes. *arXiv:1801.04255*, 2018.

[58] Daniel Litinski and Felix von Oppen. Lattice surgery with a twist: Simplifying Clifford gates of surface codes. *Quantum*, 2, 2018.

[59] Robert Raussendorf and Jim Harrington. Fault-tolerant quantum computation with high threshold in two dimensions. *Phys. Rev. Lett.*, 98:190504, May 2007.

[60] Héctor Bombín. Topological order with a twist: Ising anyons from an abelian model. *Phys. Rev. Lett.*, 105:030403, Jul 2010.

[61] Daniel Gottesman. Fault-tolerant quantum computation with higher-dimensional systems. In *Quantum Computing and Quantum Communications*, pages 302–313. Springer Berlin Heidelberg, 1999.

[62] Niel de Beaudrap and Dominic Horsman. The zx calculus is a language for surface code lattice surgery. *arXiv:1704.08670*, 2017.

[63] David Poulin. Stabilizer formalism for operator quantum error correction. *Phys. Rev. Lett.*, 95(23):230504, 2005.

[64] Christophe Vuillot, Lingling Lao, Ben Criger, Carmen García Almudéver, Koen Bertels, and Barbara M Terhal. Code deformation and lattice surgery are gauge fixing. *New Journal of Physics*, 21:033028, 2019.

[65] Earl Campbell. A theory of single-shot error correction for adversarial noise. *Quantum Science and Technology*, 2019.

[66] Omar Fawzi, Antoine Grospellier, and Anthony Leverrier. Constant overhead quantum fault-tolerance with quantum expander codes. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 743–754. IEEE, 2018.

[67] Jonas T Anderson, Guillaume Duclos-Cianci, and David Poulin. Fault-tolerant conversion between the steane and reed-muller quantum codes. *Phys. Rev. Lett.*, 113(8):080501, 2014.

[68] Kristina Renee Colladay and Erich Mueller. Rewiring stabilizer codes. *New Journal of Physics*, 2018.

[69] Sergio Boixo, Sergei V Isakov, Vadim N Smelyanskiy, Ryan Babbush, Nan Ding, Zhang Jiang, Michael J Bremner, John M Martinis, and Hartmut Neven. Characterizing quantum supremacy in near-term devices. *Nature Physics*, 14:595, 2018.

[70] Lev S Bishop, Sergey Bravyi, Andrew Cross, Jay M Gambetta, and John Smolin. Quantum volume. *Quantum Volume. Technical Report*, 2017.

[71] Norbert M Linke, Dmitri Maslov, Martin Roetteler, Shantanu Debnath, Caroline Figgatt, Kevin A Landsman, Kenneth Wright, and Christopher Monroe. Experimental comparison of two quantum computing architectures. *Proceedings of the National Academy of Sciences*, 114(13):3305–3310, 2017.

[72] Tzvetan S Metodi, Darshan D Thaker, Andrew W Cross, Frederic T Chong, and Isaac L Chuang. Scheduling physical operations in a quantum information processor. In *Quantum Information and Computation IV*, volume 6244, page 62440T. International Society for Optics and Photonics, 2006.

[73] Mark Whitney, Nemanja Isailovic, Yatish Patel, and John Kubiatowicz. Automated generation of layout and control for quantum circuits. In *Proceedings of the 4th international conference on Computing frontiers*, pages 83–94. ACM, 2007.

[74] Mohammad Javad Dousti and Massoud Pedram. Minimizing the latency of quantum circuits during mapping to the ion-trap circuit fabric. In *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 840–843. IEEE, 2012.

[75] Maryam Yazdani, Morteza Saheb Zamani, and Mehdi Sedighi. A quantum physical design flow using ilp and graph drawing. *Quantum information processing*, 12(10):3239–3264, 2013.

[76] Tayebeh Bahreini and Naser Mohammadzadeh. An minlp model for scheduling and placement of quantum circuits with a heuristic solution approach. *JETC*, 12(3):29, 2015.

[77] Aaron Lye, Robert Wille, and Rolf Drechsler. Determining the minimal number of swap gates for multi-dimensional nearest neighbor quantum circuits. In *The 20th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 178–183. IEEE, 2015.

[78] Robert Wille, Oliver Keszocze, Marcel Walter, Patrick Rohrs, Anupam Chattopadhyay, and Rolf Drechsler. Look-ahead schemes for nearest neighbor optimization of 1d and 2d quantum circuits. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 292–297. IEEE, 2016.

[79] Azim Farghadan and Naser Mohammadzadeh. Quantum circuit physical design flow for 2d nearest-neighbor architectures. *International Journal of Circuit Theory and Applications*, 45(7):989–1000, 2017.

[80] IBM. Quantum information software kit. `https://github.com/QISKit`, 2018.

[81] Alwin Zulehner, Alexandru Paler, and Robert Wille. An efficient methodology for mapping quantum circuits to the ibm QX architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[82] Marcos Yukio Siraichi, Vinícius Fernandes dos Santos, Sylvain Collange, and Fernando Magno Quintão Pereira. Qubit allocation. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, pages 113–125. ACM, 2018.

[83] Davide Venturelli, Minh Do, Eleanor Rieffel, and Jeremy Frank. Compiling quantum circuits to realistic hardware architectures using temporal planners. *Quantum Science and Technology*, 3(2):025004, 2018.

[84] Steven Balensiefer, Lucas Kreger-Stickles, and Mark Oskin. QUALE: quantum architecture layout evaluator. In *Quantum Information and Computation III*, volume 5815, pages 103–114. International Society for Optics and Photonics, 2005.

[85] Mohammad Javad Dousti and Massoud Pedram. LEQA: latency estimation for a quantum algorithm mapped to a quantum circuit fabric. In *Proceedings of the 50th Annual Design Automation Conference (DAC))*, page 42. ACM, 2013.

[86] Mohammad Javad Dousti, Alireza Shafaei, and Massoud Pedram. Squash: a scalable quantum mapper considering ancilla sharing. In *Proceedings of the 24th edition of the great lakes symposium on VLSI*, pages 117–122. ACM, 2014.

[87] Muhammad Ahsan. *Architecture Framework for Trapped-Ion Quantum Computer based on Performance Simulation Tool*. PhD thesis, Duke University, 2015.

[88] Jeff Heckey, Shruti Patil, Ali JavadiAbhari, Adam Holmes, Daniel Kudrow, Kenneth R Brown, Diana Franklin, Frederic T Chong, and Margaret Martonosi. Compiler management of communication and parallelism for quantum computation. In *Proceedings of 20th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 445–456. ACM, 2015.

[89] Chia-Chun Lin, Susmita Sur-Kolay, and Niraj K Jha. PAQCS: Physical design-aware fault-tolerant quantum circuit synthesis. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(7):1221–1234, 2014.

[90] Sergey B Bravyi and Alexei Kitaev. Quantum codes on a lattice with boundary. *arXiv:9811052*, 1998.

[91] Alexandru Paler, Simon J Devitt, and Austin G Fowler. Synthesis of arbitrary quantum circuits to topological assembly. *Scientific reports*, 6:30600, 2016.

[92] Alexandru Paler, Ilia Polian, Kae Nemoto, and Simon J Devitt. Fault-tolerant, high-level quantum circuits: form, compilation and description. *Quantum Science and Technology*, 2(2):025003, 2017.

[93] Alexandru Paler, Austin G Fowler, and Robert Wille. Online scheduled execution of quantum circuits protected by surface codes. *arXiv:1711.01385*, 2017.

[94] Ali Javadi-Abhari, Pranav Gokhale, Adam Holmes, Diana Franklin, Kenneth R Brown, Margaret Martonosi, and Frederic T Chong. Optimized surface code communication in superconducting quantum computers. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 692–705. ACM, 2017.

[95] Daniel Herr, Franco Nori, and Simon J Devitt. Optimization of lattice surgery is np-hard. *npj Quantum Information*, 3(1):35, 2017.

[96] Robert Raussendorf, Jim Harrington, and Kovid Goyal. A fault-tolerant one-way quantum computer. *Annals of physics*, 321(9):2242–2270, 2006.

[97] Robert Raussendorf, Jim Harrington, and Kovid Goyal. Topological fault-tolerance in cluster state quantum computation. *New Journal of Physics*, 9(6):199, 2007.

[98] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[99] Alireza Shafaei, Mehdi Saeedi, and Massoud Pedram. Qubit placement to minimize communication overhead in 2d quantum architectures. In *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 495–500. IEEE, 2014.

[100] Arthur Richards, Tom Schouwenaars, Jonathan P How, and Eric Feron. Spacecraft trajectory planning with avoidance constraints using mixed-integer linear programming. *Journal of Guidance, Control, and Dynamics*, 25(4):755–764, 2002.

[101] L Kaufman and Fernand Broeckx. An algorithm for the quadratic assignment problem using bender's decomposition. *EJOR*, 2(3):207–211, 1978.

[102] Chia Chun Lin, Amlan Chakrabarti, and Niraj Kumar Jha. QLib: Quantum module library. *ACM Journal on Emerging Technologies in Computing Systems*, 11(1):7, 2014.

[103] D Michael Miller, Dmitri Maslov, and Gerhard W Dueck. A transformation based algorithm for reversible logic synthesis. In *Proceedings 2003. Design Automation Conference (IEEE Cat. No. 03CH37451)*, pages 318–323. IEEE, 2003.

[104] Richard P Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21:467–488, 1982.

[105] Ivan Kassal, James D Whitfield, Alejandro Perdomo-Ortiz, Man-Hong Yung, and Alán Aspuru-Guzik. Simulating chemistry using quantum computers. *Annual Review of Physical Chemistry*, 62:185–207, 2011.

[106] Xiang Fu, Michiel Adriaan Rol, Cornelis Christiaan Bultink, J van Someren, Nader Khammassi, Imran Ashraf, R F L Vermeulen, J C de Sterke, W J Vlothuizen, R N Schouten, Carmen G Almudever, Leonardo DiCarlo, and Koen Bertels. An experimental microarchitecture for a superconducting quantum processor. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-50)*, pages 813–825. IEEE/ACM, 2017.

[107] Xiang Fu, Leon Riesebos, Michiel Adriaan rOL, Jeroen van Straten, J van Someren, Nader Khammassi, Imran Ashraf, RFL Vermeulen, V Newsum, KKL Loh, et al. eQASM: An executable quantum instruction set architecture. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 224–237. IEEE, 2019.

[108] Emanuel Knill. Quantum computing with realistically noisy devices. *Nature*, 434:39, 2005.

[109] Leon Riesebos, Xiang Fu, Savvas Varsamopoulos, Carmen G Almudever, and Koen Bertels. Pauli frames for quantum computer architectures. In *Proceedings of the 54th Annual Design Automation Conference (DAC)*, page 76. ACM, 2017.

[110] Austin G Fowler, Adam C Whiteside, and Lloyd C L Hollenberg. Towards practical classical processing for the surface code. *Phys. Rev. Lett.*, 108:180501, 2012.

[111] Giacomo Torlai and Roger G Melko. Neural decoder for topological codes. *Phys. Rev. Lett.*, 119:030501, 2017.

[112] Savvas Varsamopoulos, Koen Bertels, and Carmen G Almudever. Designing neural network based decoders for surface codes. *arXiv:1811.12456*, 2018.

[113] Yu Tomita and Krysta M Svore. Low-distance surface codes under realistic quantum noise. *Phys. Rev. A*, 90:062320, 2014.

[114] Theodore J Yoder and Isaac H Kim. The surface code with a twist. *Quantum*, 1:2, 2017.

[115] Lingling Lao, Bas van Wee, Imran Ashraf, J van Someren, Nader Khammassi, Koen Bertels, and Carmen G Almudever. Mapping of lattice surgery-based quantum circuits on surface code architectures. *Quantum Science and Technology*, 4:015005, 2019.

[116] Swamit S Tannu, Zachary A Myers, Prashant J Nair, Douglas M Carmean, and Moinuddin K Qureshi. Taming the instruction bandwidth of quantum computers via hardware-managed error correction. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-50)*, pages 679–691. IEEE/ACM, 2017.

[117] Stephane Beauregard. Circuit for Shor's algorithm using $2n + 3$ qubits. *arXiv:0205095*, 2002.

[118] Julian Kelly. A preview of Bristlecone, google's new quantum processor. News from google ai, Google LLC, 2018.

[119] Will Knight. IBM raises the bar with a 50-qubit quantum computer. News, MIT Technology Review, 2018.

[120] Jeremy Hsu. CES 2018: Intel's 49-qubit chip shoots for quantum supremacy. General technology blog, IEEE Spectrum, 2018.

[121] Cornelius Hempel, Christine Maier, Jonathan Romero, Jarrod McClean, Thomas Monz, Heng Shen, Petar Jurcevic, Ben P Lanyon, Peter Love, Ryan Babbush, et al. Quantum chemistry calculations on a trapped-ion quantum simulator. *Phys. Rev. X*, 8(3):031022, 2018.

[122] C Kokail, C Maier, R van Bijnen, T Brydges, MK Joshi, P Jurcevic, CA Muschik, P Silvi, R Blatt, CF Roos, et al. Self-verifying variational quantum simulation of lattice models. *Nature*, 569(7756):355, 2019.

[123] Yunong Shi, Nelson Leung, Pranav Gokhale, Zane Rossi, David I Schuster, Henry Hoffmann, and Frederic T Chong. Optimized compilation of aggregated instructions for realistic quantum computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1031–1044. ACM, 2019.

[124] Peter W Shor. Fault-tolerant quantum computation. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 56–65. IEEE, 1996.

[125] Andrew M Steane. Active stabilization, quantum computation, and quantum state synthesis. *Phys. Rev. Lett.*, 78(11):2252, 1997.

[126] Emanuel Knill. Scalable quantum computing in the presence of large detected-error rates. *Phys. Rev. A*, 71(4):042322, 2005.

[127] Rui Chao and Ben W Reichardt. Quantum error correction with only two extra qubits. *Phys. Rev. Lett.*, 121(5):050502, 2018.

[128] Christopher Chamberland and Michael E Beverland. Flag fault-tolerant error correction with arbitrary distance codes. *Quantum*, 2:53, 2018.

[129] Ben W Reichardt. Fault-tolerant quantum error correction for steane's seven-qubit color code with few or no extra qubits. *arXiv:1804.06995*, 2018.

[130] Rigetti. Regetti forest. `https://www.rigetti.com/forest`, 2018.

[131] David P DiVincenzo and Panos Aliferis. Effective fault-tolerant quantum computation with slow measurements. *Phys. Rev. Lett.*, 98(2):020501, 2007.

[132] Austin G Fowler. Minimum weight perfect matching of fault-tolerant topological quantum error correction in average o (1) parallel time. *Quantum Information & Computation*, 15(1-2):145–158, 2015.

[133] Guillaume Duclos-Cianci and David Poulin. Fast decoders for topological quantum codes. *Phys. Rev. Lett.*, 104(5):050504, 2010.

[134] Stefan Krastanov and Liang Jiang. Deep neural network probabilistic decoder for stabilizer codes. *Scientific reports*, 7(1):11003, 2017.

[135] Paul Baireuther, Thomas E O'Brien, Brian Tarasinski, and Carlo WJ Beenakker. Machine-learning-assisted correction of correlated qubit errors in a topological code. *Quantum*, 2:48, 2018.

[136] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[137] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[138] Rodney Van Meter and Clare Horsman. A blueprint for building a quantum computer. *Communications of the ACM*, 56(10):84–93, 2013.

[139] Andrew W Cross, Lev S Bishop, Sarah Sheldon, Paul D Nation, and Jay M Gambetta. Validating quantum computers using randomized model circuits. *arXiv:1811.12926*, 2018.

[140] Morten Kjaergaard, Mollie E Schwartz, Jochen Braumüller, Philip Krantz, Joel I-Jan Wang, Simon Gustavsson, and William D Oliver. Superconducting qubits: Current state of play. *arXiv:1905.13641*, 2019.

[141] SA Caldwell, N Didier, CA Ryan, EA Sete, A Hudson, P Karalekas, R Manenti, MP da Silva, R Sinclair, E Acala, et al. Parametrically activated entangling gates using transmon qubits. *Phys. Rev. Applied*, 10(3):034050, 2018.

[142] Serwan Asaad, Christian Dickel, Nathan K Langford, Stefano Poletto, Alessandro Bruno, Michiel Adriaan Rol, Duije Deurloo, and Leonardo DiCarlo. Independent, extensible control of same-frequency superconducting qubits by selective broadcasting. *npj Quantum Information*, 2:16029, 2016.

[143] David C McKay, Thomas Alexander, Luciano Bello, Michael J Biercuk, Lev Bishop, Jiayin Chen, Jerry M Chow, Antonio D Córcoles, Daniel Egger, Stefan Filipp, et al. Qiskit backend specifications for openqasm and openpulse experiments. *arXiv:1809.03452*, 2018.

[144] Steven Herbert and Akash Sengupta. Using reinforcement learning to find efficient qubit routing policies for deployment in near-term quantum computers. *arXiv:1812.11619*, 2018.

[145] Will Finigan, Michael Cubeddu, Thomas Lively, Johannes Flick, and Prineha Narang. Qubit allocation for noisy intermediate-scale quantum computers. *arXiv:1810.08291*, 2018.

[146] Gushu Li, Yufei Ding, and Yuan Xie. Tackling the qubit mapping problem for nisq-era quantum devices. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1001–1014. ACM, 2019.

[147] Swamit S Tannu and Moinuddin K Qureshi. Not all qubits are created equal: A case for variability-aware policies for nisq-era quantum computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 987–999. ACM, 2019.

[148] Shin Nishio, Yulu Pan, Takahiko Satoh, Hideharu Amano, and Rodney Van Meter. Extracting success from ibm's 20-qubit machines using error-aware compilation. *arXiv:1903.10963*, 2019.

[149] Alexander Cowtan, Silas Dilkes, Ross Duncan, Alexandre Krajenbrink, Will Simmons, and Seyon Sivarajah. On the qubit routing problem. *arXiv:1902.08091*, 2019.

[150] QuTech. OpenQL Compiler. `https://github.com/QE-Lab/OpenQL`, 2019.

[151] Nader Khammassi, Gian G Guerreschi, Imran Ashraf, Justin W Hogaboam, Carmen G Almudever, and Koen Bertels. cQASM v1. 0: Towards a common quantum assembly language. *arXiv:1805.09607*, 2018.

[152] Robert Wille, Daniel Große, Lisa Teuber, Gerhard W Dueck, and Rolf Drechsler. Revlib: An online resource for reversible functions and reversible circuits. In *38th International Symposium on Multiple Valued Logic (ismvl 2008)*, pages 220–225. IEEE, 2008.

[153] Nader Khammassi, Imran Ashraf, Xiang Fu, Carmen G Almudéver, and Koen Bertels. QX: A high-performance quantum computer simulation platform. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 464–469. IEEE, 2017.

[154] Prakash Murali, Norbert Matthias Linke, Margaret Martonosi, Ali Javadi Abhari, Nhung Hong Nguyen, and Cinthia Huerta Alderete. Full-stack, real-system quantum computer studies: Architectural comparisons and design insights. *arXiv:1905.11349*, 2019.

[155] Prakash Murali, Jonathan M Baker, Ali Javadi Abhari, Frederic T Chong, and Margaret Martonosi. Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers. *arXiv:1901.11054*, 2019.

[156] Davide Venturelli, Minh Do, Bryan O'Gorman, Jeremy Frank, Eleanor Rieffel, Kyle EC Booth, Thanh Nguyen, Parvathi Narayan, and Sasha Nanda. Quantum circuit compilation: An emerging application for automated reasoning. 2019.

[157] Intel. Intel newsroom. `https://newsroom.intel.com/press-kits/quantum-computing/#intel-qutech`, 2019.

[158] Easwar Magesan, Daniel Puzzuoli, Christopher E Granade, and David G Cory. Modeling quantum noise for efficient testing of fault-tolerant circuits. *Phys. Rev. A*, 87(1):012324, 2013.

[159] Jeff P Barnes, Colin J Trout, Dennis Lucarelli, and BD Clader. Quantum error-correction failure distributions: Comparison of coherent and stochastic error models. *Phys. Rev. A*, 95(6):062338, 2017.

[160] Andrew S Darmawan and David Poulin. Tensor-network simulations of the surface code under realistic noise. *Phys. Rev. Lett.*, 119(4):040502, 2017.

[161] Sergey Bravyi, Matthias Englbrecht, Robert König, and Nolan Peard. Correcting coherent errors with surface codes. *npj Quantum Information*, 4(1):55, 2018.

[162] Daniel Litinski. A game of surface codes: Large-scale quantum computing with lattice surgery. *Quantum*, 3:128, 2019.

[163] Helmut G Katzgraber, Héctor Bombín, Ruben S Andrist, and Miguel A Martin-Delgado. Topological color codes on union jack lattices: a stable implementation of the whole Clifford group. *Phys. Rev. A*, 81(1):012319, 2010.

[164] Aleksander Kubica and Michael E Beverland. Universal transversal gates with color codes: A simplified approach. *Phys. Rev. A*, 91(3):032330, 2015.

[165] Héctor Bombín and Miguel A Martin-Delgado. Topological computation without braiding. *Phys. Rev. Lett.*, 98:160502, 2007.

[166] Aleksander Kubica, Beni Yoshida, and Fernando Pastawski. Unfolding the color code. *New Journal of Physics*, 17(8):083026, 2015.

[167] Thomas Camara, Harold Ollivier, and J-P Tillich. Constructions and performance of classes of quantum LDPC codes. *arXiv:0502086*, 2005.

[168] Matthew B Hastings. Decoding in hyperbolic spaces: LDPC codes with linear rate and efficient error correction. *arXiv:1312.2546*, 2013.

[169] Jean-Pierre Tillich and Gilles Zémor. Quantum LDPC codes with positive rate and minimum distance proportional to the square root of the blocklength. *IEEE Transactions on Information Theory*, 60(2):1193–1202, 2013.

ACKNOWLEDGEMENTS

The life of a PhD student is very much like the weather in the Netherlands. For most of the time, it is cloudy, windy and rainy, but that may be the reason why we feel so happy when the sunshine comes. At this sunny moment of my life, I would like to acknowledge all the people who have helped me during this long journey in one way or another.

Firstly, I would like to express my sincere gratitude to my promoter Prof. Koen Bertels. Dear **Koen**, thank you for giving me the opportunity to pursue a doctoral degree in quantum computing in Delft. I am very grateful for your guidance in my research topics, for your efforts and patience in training and developing my soft skills, for your support and encouragement. Also (as all the other PhDs in this group), thank you for all the nice conversations and fun during the pizza-wine-beer-soda nights, BBQs, Carting, Bowling, etc.

Secondly, I want to show my gratitude to my supervisor, Dr. Carmen G. Almudever. I sincerely appreciate the time you spent discussing ideas, helping me to prepare talks, reviewing papers. I still remember that you went through the slides sentence by sentence with me for my first presentation at FT Werkbespreking. I am also thankful for your kindness. You are always so considerate and are like a big sister who looks after me. Dear **Carmina**, thank you for your patience, support, and protection in this journey.

Next, I would like to convey my thanks to Dr. Daniel B. Criger. **Ben**, you are the one who has guided me in the world of quantum error correction and told me to be critical and rigorous in research. Without you, I would not have learned so much. Thank you so much for your supervision, your enlightenment, and your support.

Special thanks to my committee: **Prof. Lieven Vandersypen, Prof. Robert König, Prof. Harry Buhrman, Prof. Lenardo DiCarlo, Prof. Barbara Terhal**, for reading this thesis and giving me useful comments to improve it. Especially, I want to extend my gratitude to Barbara, thank you for the collaboration and the advices which will benefit me for the rest of my life.

I would also like to express my appreciations to my colleagues who have made my PhD life enjoyable and unforgettable.

First, I want to thank my colleagues in the QCA Lab. **Xiang**, thank you for the countless discussions and conversations during our PhD. You are not only a colleague, but also a friend whom I can share thoughts with, a big brother who has been taking care of me, I feel lucky to meet you in this group. As you wrote in your thesis, I also wish 'we can continue our friendship and collaboration in the rest of our lives'. **Leon**, you are so sweet. It was a happy thing to drop by your office everyday and have a nice chat. Thanks for organizing so many social activities and bringing me a lot of fun. **Hans**, you are always willing to share your experience and help all the students, thank you for all the brainstorming and discussions in the circuit mapping problem. Also, thanks for translating my propositions into dutch. **Imran**, you are so kind, humble, and helpful. Thank you for helping me with all the software problems. **Savvas**, the handsome Greek officemate, thanks a lot

for all the delightful conversations and the advices on Greek islands, we enjoyed Crete a lot. My other colleagues in QCA, **Nader, Dan, Andreas, Aritra, Abid, Mengyu, Miguel, Diogo, Amitabh, Jeroen, Matthijs, Medina**, thank you for your support, interesting discussions, and fun during lunches, dinners, and drinks. I also want to thank my students **Daniel, Alex, Yaoling, Peter-Jan** for your trust. I have also learned a lot from you, the young and creative guys. **Peter-Jan**, thank you for translating the summary of my thesis into Dutch, it must take a lot of efforts.

Thanks to my other colleagues at the department of quantum and computer engineering (QCE). **Joyce** and **Lidwina**, you are the best secretaries I have met so for. You were so nice and patient, thank you for helping me with all the administrative problems and for organizing so many fun Ladies' activities. Thanks to **Erik** for your assistance with the use of servers. The CE.cn guys, **Lei, Jintao, Shanshan, Pengfei, Lizhou, Yande, He, Baozhou**, thank you for all your help and all the pleasurable Lunches together. **Anh, Hale, Mahroo, Cucu, Arwa, Trisha, Laura, Said, Zaid, Motta, Stephan, Sorin, Troya, Haji, Daniel, Moritz, Abdullah, Cesar, Gerd, Ramon, Lizzy, Qiang, Long**, and other QCEers, thanks for all the chats, drinks, and joy.

QuTech is a fascinating place because of its remarkable good work and incredible lively environment from hard-working, brilliant, and creative people. Xiaotong, Nelly, Lin, Jie, Yuanxing, Yang, Chunxiao, Xiao, Qingzhen, Di, it is so pleasant to talk quantum in Chinese in Delft. **Nelly, Lin, Jie**, you have motivated me a lot in different ways, which is a precious gift (probably you do not even know). I am so happy to meet you and know you. **Xiaotong**, many thanks for your help and useful suggestions, for pushing me and encouraging me to talk with others during conferences, and special thanks to your wife's delicious dumplings which made me feel at home. **Yuanxing, Yang, Chunxiao, Xiao, Qingzhen, Di**, thank you for sharing experience and ideas, and for bringing fun during all the dinners and parties. **Christophe**, you are a kind person and a talented scientist, I really appreciate the collaboration with you. **Menno, Fabio, Francesco, Ariaan, Daniel, Jonathan, Jonas, Mark**, and many QuTechers, thank you so much for your help and your scholarly interactions. My deep thanks to the golden Uitje 2019 committee: **Arian, Arno, Gertjan, Josh, Vanessa**, thanks for making such a wonderful and special memory, for your hard work, and for all the fun together. It was a pleasure to work with you.

Now, I would like to show my gratitude to my dear friends in the Netherlands and in China. The werewolf group, **Yu, Zhijie, Tiantian, Jian, Zhaokun, Xiang, Yueyue, ZiXuan**, thank you for being with me for these four years. I miss the time we spent together so much drinking beers and wines, cooking Chinese dishes, playing board games, having trips. My dear **Yu**, we baked cakes for each other, we shared good wines together, we exchanged life experience and supported each other, I am so happy to have a sweet friend like you in the bitter PhD life. **Zhijie**, we all love your innocent smiles, thank you for taking care of my Yu :). **Jian**, you are such an excellent cook, though you did not cook that much for us any more (you know what I mean, right?). **Yueyue**, your optimistic personality makes you a light of this group, it was always comforting and relaxing to have you around. The beautiful girls, **Tiantian, Hongjuan, Renfei, Mingjuan**, it is great to meet you and go to Zumba classes together on Tuesdays and Thursdays. Thank you for your listening, sharing, and encouragement during the hard time of my PhD. **Tiantian**, thank you for helping me to design and draw the cover using the painful tool, Adobe Illustra-

tor :). Thanks to **Qin, Shengzhi**, and many other friends in Delft for all your help and accompany.

True friends will not be separated by the physical distance, I believe this because of you, my dear friends in China. I want to thank you, too many memories came to my mind and I could not find the words to express myself. **Jina Dong, Biao Liu**, you were there whenever I needed you, you will be always there. The Beijing drifters, **Dandan Ge, Jiaming Xu, Dan Wu, Meng Xiao, Wei Qin**, every time when you spent more than two hours for meeting me, I wanted to cry. Let us keep the reunions going on for the rest of our lives. One of the reasons that I love HIT is because I met you there, **Yulu Zhang, Zongxun Zhang, Dan Gu, Feifei**, I am so grateful to have you. **Meng Zhu, Guanglong Zhang, Lujie Wang, Hao Sun, Xun Duan, Kai Jing**, thank you for hosting me whenever I go back to Xi'an.

Finally, my deepest gratitude goes to my family. Mama (Yanhua Liu), I wish you would be proud of me if you could know that I will become a doctor. Papa (Guiyou Lao), I owe you too much. We were going through so much together, I could not ever imagine the difficulties for you to raise me up. You give me unconditional love and support me to accomplish my goals. I would be nothing without you and my gratitude is beyond the word's description. Mum (Caixia Jing) and Dad (Yongan Zheng), I still remember on the day Zixuan and I registered to be couples, you said 'Lingling, you are our little daughter (not only in law)'. You have been taking care of me, loving me, and supporting me as a mother and a father (not only in law). I feel deeply indebted to you and I do not know how to thank you enough.

Last but not least, I want to thank my beloved husband, Zixuan. Here, in the Netherlands, we built our first home, bought our first car, traveled around the world. We started our PhD life, helped, encouraged, and supported each other. I have been used to being protected and looked after by you, it is like a habit, maybe even an addiction. Without you, I might not survive from the pain of pursuing a doctorate (1.5 :)). You have made and will continue to make my life beautiful, enjoyable, and special. 'You are (also) the best thing about me'.

Lingling
November, 2019
Delft, the Netherlands

CURRICULUM VITÆ

Lingling LAO

06-12-1989 Born in Qiqihar, Heilongjiang Province, China.

EDUCATION

2008–2012 Undergraduate, School of Electronics and Information Engineering
Harbin Institute of Technology
Harbin, China

2012–2014 Graduate, School of Electronics and Information
Northwestern Polytechnical University
Xi'an, China

2015–2019 PhD, QuTech
Delft University of Technology
Delft, The Netherlands

List of Publications

Journal Papers

1. **L. Lao**, B. van Wee, I. Ashraf, J. van Someren, N. Khammassi, K. Bertels, C. G. Almudever. *Mapping Lattice Surgery-based Quantum Circuits onto Surface Code Architectures*, Quantum Science and Technology **4(1)**, 015005 (2018).

2. C. Vuillot*, **L. Lao***(share first authorship), B. Criger, C. G. Almudever, K. Bertels, B. Terhal. *Code Deformation and Lattice Surgery are Gauge Fixing*, New Journal of Physics **21**, 033028 (2019).

3. X. Fu*, **L. Lao***(share first authorship), K. Bertels, C. G. Almudever. *A Control Microarchitecture for Fault-tolerant Quantum Computing*, Microprocessors and Microsystems **70**, 21-30 (2019).

Conference Proceedings

1. L. Riesebos, X. Fu, A. A. Moueddenne, **L. Lao**, S. Varsamopoulos, I. Ashraf, J. van Someren, N Khammassi, C. G. Almudever, K Bertels. *Quantum Accelerated Computer Architectures*, in Proceedings of the IEEE International Symposium on Circuits and Systems (2019).

2. C. G. Almudever, **L. Lao**, X. Fu, N. Khammassi, I. Ashraf, D. Iorga, S. Varsamopoulos, C. Eichler, A. Wallraff, L. Geck, A. Kruth, J. Knoch, H. Bluhm, K. Bertels. *The Engineering Challenges in Quantum Computing*, in Design, Automation & Test in Europe Conference & Exhibition (2017).

3. X. Fu, L. Riesebos, **L. Lao**, C. G. Almudever, F. Sebastiano, R. Versluis, E. Charbon, K. Bertels. . *A Heterogeneous Quantum Computer Architecture*, in Proceedings of the ACM International Conference on Computing Frontiers (2016).

Preprints

1. **L. Lao**, C. G. Almudever. *Fault-tolerant Quantum Error Correction on Near-term Processor using Flag and Bridge Qubits*, arXiv:1909.07628 (2019).

2. **L. Lao**, D. M.Manzano, J. van Someren, I. Ashraf, C. G. Almudever. *Mapping of Quantum Circuits onto NISQ Superconducting Processors*, arXiv:1908.04226 (2019).

ISBN:978-94-028-1838-3