THE UNIVERSITY OF CHICAGO

PRACTICAL VARIATION-AWARE DESIGNS IN QUANTUM COMPUTING

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY
SOPHIA LIN

CHICAGO, ILLINOIS
AUGUST 2024

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Throughout my PhD journey I have been fortunate to meet many people who have offered encouragement, valuable advice, and inspiration. First I would like to thank my advisor Fred Chong for his guidance and support throughout my PhD. I would also like to thank my committee members Ben Brown and Bill Fefferman for their time. My three summer internships were enriching experiences, thanks to the mentorship of Dan, Andrew, Drew, Eric, Prashant, and Krishanu. I am grateful to my labmates and other collaborators, including Andy, Casey, Charles, Charlie, Chris, Gokul, Jason, Jonathan, Josh, Kate, Max, Michael, Natalia, Pranav, Ryan, Sam, Sara, Siddarth, Teague, Wei, Yongshan, and Yunong. Lastly, I am grateful for the support of my family and friends.

ABSTRACT

Variations are prevalent in all aspects of quantum computing. On solid state quantum devices, fabrication errors lead to variations in device connectivity. Among the qubits that are available for use, there are still variations in multiple properties. Other than hardware variations, different algorithms and operations impose different requirements on the devices and systems. In order to bridge the gap between the theory and implementation of quantum computing, we need practical designs that are aware of variations and system-level tradeoffs. This thesis includes three examples of adapting to variations: choosing two-qubit basis gates based on individual qubits' properties, adapting error correction codes and using modular architecture to support fault-tolerant computation in the presence of fabrication defects, and adapting real time decoding protocols to support large patches of topological codes that arise during lattice surgery operations.

CHAPTER 1

INTRODUCTION

Quantum computing is an interdisciplinary field that encompasses diverse research areas. Its advancement requires both highly abstract theory research, like the high-level design of quantum algorithms and error correction codes, and experimental efforts that focus on the development and implementation of quantum hardware. Theory works make simplifying assumptions about the underlying hardware, which allow them to focus on the core concepts. For example, it is common for a paper that proposes a novel quantum algorithm to assume that it runs on an ideal noiseless quantum device, and for a paper on quantum error correction to use a simplified phenomenological noise model. In addition to ignoring noises, another common assumption is uniformity, including but not limited to uniformity across physical qubits on a device.

While simplification is necessary for research, various complications arise when it comes to practically implementing quantum applications on hardware. Qubits are not only susceptible to a variety of noises that require mitigation, but some of them, including the superconducting qubits that are the currently leading candidates, also exhibit inhomogeneity. This leads to variability in performance and characteristics, even among qubits on the same device. In the worst case, a qubit or link between qubits could have severely limited functionality, reducing the connectivity of the device. The design of quantum computing architectures must be flexible enough to accommodate these variations in qubit behavior and performance.

Research in software and computer architecture should also meet the various demands imposed by quantum algorithms. For example, fault-tolerant quantum algorithms require real time decoding of syndromes. For superconducting devices with fast syndrome measurement cycles, it is hard for a software decoder designed to run on a CPU to meet the requirement for real time decoding. In recent years, there have been real time decoders based

on FPGAs, ASICs, or SFQ circuits. Different quantum error correction codes also impose distinct requirements on quantum computers. Topological codes like the surface code and the color code only require nearest-neighbor interactions, but their low encoding rates leads to a high qubit overhead. On the contrary, some qLDPC codes have high encoding rates but require long range connectivity.

This dissertation presents three examples of adapting to variations that are often overlooked. Specifically, Chapter 2 and 3 address different aspects of the variations in qubit characteristics, and Chapter 4 involves a variation in quantum application.

Chapter 2 introduces a framework for selecting different "nonstandard" two-qubit basis gates for each link on a superconducting quantum device. Conventionally, it is assumed that the same two-qubit basis gate (typically an XX- or XY-type gate) is implemented between every pair of neighboring qubits on a quantum device. This assumption is prevalent in quantum compiler research, and the research that studies the expressivity of different two-qubit gates. However, if the same target two-qubit gate is selected as basis gate for the entire device, even a small amount of coherent crosstalk between two qubits will cause a gate to have a deviation from the target. We instead studied how to select the most efficient "nonstandard" two-qubit basis gate, given the systematic deviations in the Cartan trajectory of an interaction.

Chapter 3 considers the problem of implementing quantum error correction codes on defective arrays of qubits. Prior work had proposed techniques that deforms the surface code in the presence of defective qubits and links. We proposed to implement surface codes on modular chiplets, and use the flexibility of a modular design to mitigate the resource overhead caused by fabrication defects. Our work addressed several practical challenges that need to be solved before one can build a large scale fault-tolerant device that implements surface code. We developed an automated protocol for adapting the rotated surface code to a grid with an arbitrary defect distribution. With this tool, we performed numerical

simulations to identify the key metrics that characterize the fidelity of a surface code on a defective array of qubits. These metrics serve a dual purpose: firstly, as post-selection criteria for the strategic selection and arrangement of chiplets; secondly, they facilitate the simulations to assess the resource overheads. We also determine cutoff fidelity values that help identify whether a qubit should be disabled or kept as part of the error correction code.

The topic in Chapter 4 is adapting real time decoders to the large patches of codes that arise during lattice surgery. This is accomplished by dividing the device into multiple overlapping windows and assigning a decoder module to each window. We refer to this approach as spatially parallel windows. While previous work has explored similar ideas, none have addressed the unique hardware constraints and system-specific considerations pertinent to the task. We show how to configure spatially parallel windows, so that the scheme (1) is compatible with hardware accelerators, (2) supports general lattice surgery operations, (3) maintains the fidelity of the logical qubits, and (4) meets the throughput requirement for real time decoding. Furthermore, our results reveal the importance of optimally choosing the buffer width to achieve a balance between accuracy and throughput — a decision that should be influenced by the device's physical noise.

These examples demonstrate the need for quantum computer architecture research that bridges theory and hardware, both for physical operations and logical operations. Chapter 5 concludes with a discussion of future work in this direction.

CHAPTER 2

NONSTANDARD TWO-QUBIT BASIS GATES FOR QUBIT VARIATIONS

2.1 Introduction

Quantum computers have the potential to solve problems currently intractable for conventional computers [107], but current computations are limited by errors [99], particularly when interacting two qubits to perform a quantum gate operation. This is not surprising, as qubits are engineered to preserve quantum state and be isolated from the environment, but a quantum operation is a moment in time where external control is applied from the environment to deliberately alter a qubit's state. To accomplish low-error gates, the control mechanisms are carefully designed and the control signals are calibrated for each qubit or pair of qubits.

Similar to how classical computers use a small set of classical logic gates (AND, OR, NOT, XOR...) as building blocks for larger circuits, current superconducting quantum devices typically only directly support a universal gate set consisting of a few two-qubit (2Q) gates and a continuous set of single-qubit (1Q) gates. This paper will refer to the set of directly supported quantum gates as *basis gates*. In the space of 2Q gates (see Fig.2.1), any point that does not coincide with SWAP or Identity has nonzero entangling power. Any of these 2Q entangling gates can achieve universal computation when added to a continuous set of 1Q gates [20].

Using the minimal set of gates needed for universal computing is rarely a desirable thing to do. For example, while the NAND is universal in classical computing, building circuits from it alone is less efficient than using a larger set of logic gates. However, the intensive calibrations necessary for high fidelity 2Q gates between qubits in a large quantum computer make it impractical to support a large set of 2Q basis gates. All logical 2Q gates scheduled to

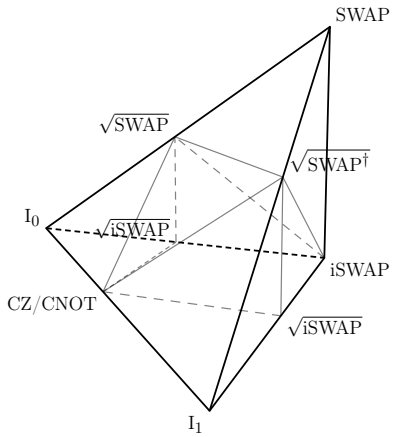


Figure 2.1: The Weyl chamber of 2Q quantum gates, explained in Sec 2.2.2. The non-local part of a 2Q gate is fully described by its position in the Weyl chamber. As the duration of an entangling gate pulse increases, the 2Q gate evolves, traversing a Cartan trajectory in the Weyl chamber. CNOT and CZ are both represented by $(\frac{1}{2}, 0, 0)$. The SWAP gate is at the top vertex $(\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$. On the bottom surface, $(t_x, t_y, 0)$ and $(1 - t_x, t_y, 0)$ represent the same equivalent class of gates. For example, the two points $I_0 = (0, 0, 0)$ and $I_1 = (1, 0, 0)$ both represent the 2Q identity gate I.

run on a quantum computer have to be decomposed by its compiler into alternating layers of pre-calibrated 1Q and 2Q basis gates. Thus, the choice of which 2Q gates to directly support is critical to enabling high-performance quantum computing. On the hardware side, the 2Q basis gates must have high-fidelity hardware implementations. On the software side, they must enable the low-depth decomposition of other 2Q gates.

Superconducting qubits support XX- and XY-type 2Q interactions [6, 72]. The strength of each of these interactions depends on the type of coupling, the coupling strength, and the frequency detuning between the qubits [72]. The Weyl chamber space of 2Q gates (Fig. 2.1) is a useful way to visualize these interactions: the coordinates of a gate correspond to its non-local part in Cartan's KAK decomposition (see Section 2.2.2). In the Weyl chamber, gates in the XX family form a straight trajectory from Identity to CNOT/CZ, while gates in the XY family form a trajectory from Identity to iSWAP. Cartan trajectories are generated by increasing the duration of an entangling gate pulse, which evolves the 2Q gate.

Section 2.4 describes the various difficulties associated with reliably performing standard

2Q gates like CZ and iSWAP on today's superconducting quantum computers. Whether a deviation from a standard Cartan trajectory is a 2Q error depends on what the target 2Q gate is. If the target 2Q gate has to be a certain standard gate (e.g. iSWAP), even a small amount of coherent crosstalk between the two qubits will cause the gate to have a small CZ component, and so the gate will not be identically iSWAP. However, if that coherent crosstalk is a stable systematic that does not add noise or cause decoherence, the gate could still be an effective, high fidelity entangler that is useful for computing; the target 2Q gate would just have to be a nonstandard unitary rather than the standard iSWAP. In Section 2.4 we also show an example of an experimentally measured Cartan trajectory of 2Q gates on a superconducting qubit device. The measured Cartan trajectory includes very fast nonstandard 2Q gates with high entangling power, but it does not pass through traditional basis gates like iSWAP and CZ. This example motivates us to develop methods that enable the use of nonstandard 2Q gates for quantum computing. If we allow for deviations from standard 2Q gates, we can use high fidelity, non-standard quantum hardware for practical quantum computing.

Using nonstandard 2Q basis gates requires methods for identifying good basis gates on a general 2Q gate trajectory, calibrating a nonstandard gate, and compiling with nonstandard basis gates. The primary focus of our work is to construct and demonstrate a framework for efficiently identifying a "good" set of 2Q basis gates from a nonstandard trajectory. But we also propose solutions for calibration and compilation.

What are our standards for a good set of 2Q basis gates? Following the principle of Amdahl's Law, we pay most attention to the SWAP gate as a target for synthesis because of its importance for communication within programs executing on superconducting devices. To mitigate crosstalk and satisfy other hardware constraints, superconducting devices usually have the sparse connectivity of a grid lattice or a hexagonal lattice. Therefore, the compiler has to schedule a series of SWAP gates before it can interact two qubits that are not adjacent

to each other. Although clever mapping from logical to physical qubits can result in a smaller number of inserted SWAP gates, we still observe a high proportion of SWAP gates in post-mapping quantum circuits. Besides efficient synthesis of the SWAP gate, our framework also allows one to prioritize other target gates, including but not limited to CNOT, iSWAP, and the B gate. It also enables the simultaneous prioritisation of multiple target gates.

The calibration of a non-standard 2Q basis gate requires identifying a gate duration that gives an ideal basis gate and then accurately characterizing the corresponding gate so that we use the right unitary for compiling. Our proposed calibration protocol address both without causing a long downtime on a quantum device. However, we point out that in order to *precisely* characterize a non-standard gate, one should consider using gate set tomography (GST) as opposed to quantum process tomography (QPT). The data collected from GST experiments may require several hours of classical processing. Before that finishes, one would have to use the calibration results from the previous cycle. The speedup of GST's classical processing, which is an active field of research [4], would help reduce the cost of calibrating non-standard gates. In addition, we observed that the systematic deviations are stable over days (Fig. 2.5). If the change in deviation is negligible, one may not need to apply GST in every calibration cycle.

Compiling with non-standard 2Q basis gates requires a conversion from arbitrary 2Q gates into the basis gates. There isn't a general analytical formula that works for arbitrary target and basis gate, so a numerical search is needed. However, we can analytically obtain information on the minimum circuit depth needed for a perfect synthesis and use it to facilitate the numerical search. Besides, the circuits that synthesise common gates from the basis gates can be pre-computed after each calibration cycle, so that one wouldn't need to re-compute them for every program.

Our contributions are summarized below.

- Our work is the first to consider using 2Q basis gates from general non-standard gate

trajectories that are not parametrized by a simple function.

- We provide a theoretical framework for identifying and visualizing the set of good 2Q basis gates, given a set of target 2Q gates to prioritize. With an emphasis on SWAP, we characterize the sets of gates that enable the synthesis of SWAP in 1, 2, and 3 layers, respectively. As another example, we visualize the gates that are able to both synthesize SWAP in 3 layers and CNOT in 2 layers. After identifying the volume of desirable basis gates in the Weyl chamber, one can select the first intersection of the trajectory with the volume as the 2Q basis gate. (Section 2.5)

- We propose a practical calibration protocol that is agnostic as to whether a 2Q gate is standard or non-standard. (Section 2.6)

- We discuss a practical approach to compiling with non-standard 2Q basis gates. (Section 2.7)

- We apply our methods to a case study entangling gate architecture with far-detuned transmon qubits [96]. First, we use our theoretical framework to select 2Q basis gates from simulated nonstandard Cartan trajectories that are realistic for this case study architecture. By increasing the entangling pulse drive amplitude we get a significant 2Q basis gate speedup but introduce a deviation into the Cartan trajectory. Then we use these 2Q basis gates to run a variety of benchmark circuits including BV[12], QAOA[38], the QFT adder[103], and the Cuccaro Adder[28], and compare to the results from using the $\sqrt{iSWAP}$ gate on the standard XY-type trajectory. (Section 2.8)

2.2 Background

2.2.1 Qubits and gates

Unlike a classical bit that is either 0 or 1, a quantum bit (qubit) can exist in a linear superposition of $|0\rangle$ and $|1\rangle$; A general quantum state can be expressed as $\alpha|0\rangle + \beta|1\rangle$ where α, β are complex amplitudes that satisfy $|\alpha|^2 + |\beta|^2 = 1$. Thus, the state of one qubit can be represented by a 2-vector of the amplitudes α and β. A system of n qubits can exist in a superposition of up to 2^n basis states, and its state can be represented by a 2^n-vector of complex amplitudes. A quantum gate that acts on n qubits can be represented by a $2^n \times 2^n$ unitary matrix.

2.2.2 Geometric characterization of 2Q gates

Two 2Q quantum gates $U_1, U_2 \in SU(4)$ are *locally equivalent* if it is possible to obtain one from the other by adding 1Q operations. In other words, 2Q operations U_1 and U_2 are locally equivalent if there exist $k_1, k_2 \in SU(2) \otimes SU(2)$ such that $U_1 = k_1 U_2 k_2$. For example, CNOT and CZ are locally equivalent via Hadamard gates.

Any 2Q quantum gate $U \in SU(4)$ can be written in the form of

$$U = k_1 \exp(-i\frac{\pi}{2}(t_x X \otimes X + t_y Y \otimes Y + t_z Z \otimes Z))k_2 \tag{2.1}$$

where X, Y, Z are the Pauli gates. This is called the Cartan decomposition.

The space of two-qubit quantum gates can be represented geometrically in a Weyl chamber (Fig. 2.1), where each point stands for a set of gates that are locally equivalent to each other [136]. The Cartan coordinates (t_x, t_y, t_z) in Eq. (2.1) are the coordinates of U in the Weyl chamber. They fully characterize the *non-local* part of a 2Q gate. On the bottom surface, $(t_x, t_y, 0)$ and $(1 - t_x, t_y, 0)$ represent the same equivalent class of gates. The other

points in the Weyl chamber each represent a different equivalence class of 2Q gates. We refer the interested readers to [27] for a more thorough introduction to the Weyl chamber. Note that other conventions of the Cartan coordinates are also common. They usually differ from ours by a constant factor of π or 2π.

In this paper, when we talk about some gate G in the Weyl chamber, we usually mean the local equivalence class of 2Q gates that includes G.

2.2.3 Entangling power of 2Q gates

The entangling power [135] is a widely accepted quantitative measure of the capacity of a 2Q gate to entangle the qubits that it acts on. It is typically a good indicator of the ability of a specific 2Q gate to synthesize arbitrary 2Q gates. For a unitary operator U, the entangling power $e_p(U) \in [0, \frac{2}{9}]$ is defined as the average linear entropy of the states produced by U acting on the manifold of all separable states [135]. It is solely based on the non-local part of U, which is characterized by the position of U in the Weyl chamber.

A 2Q gate has 0 entangling power if and only if it is locally equivalent to the Identity or the SWAP gate. Conversely, 2Q gate U is called a *perfect entangler* if it can produce a maximally entangled state from an unentangled one[136]. Perfect entanglers (PE) have entangling power no less than $\frac{1}{6}$. They constitute a polyhedron in the Weyl chamber that is exactly half of the total volume. The 6 vertices of the PE polyhedron are CZ(CNOT), iSWAP, $\sqrt{SWAP}$, $\sqrt{SWAP}^\dagger$, and the 2 points that both represent $\sqrt{iSWAP}$. The perfect entanglers with maximal entangling power of $\frac{2}{9}$ are also called *special perfect entanglers*[102]. In the Weyl chamber, they are on the line segment from CNOT to iSWAP. The B gate (Berkeley gate), which is at the midpoint of this line segment, has the property that it can synthesize any arbitrary 2Q gates within 2 layers[137]. However, there has been no proposal to directly implement the B gate in hardware.

2.3 Related work

To the best of our knowledge, no prior work involves using 2Q basis gates from arbitrary nonstandard gate trajectories. In parallel with this work, Lao et al. [73] propose to mitigate coherent parasitic errors in 2Q gates by software and present methods of compilation. Our work is more general then [73], although we share the insight that coherent errors in 2Q gates can be treated as part of the gate for compilation. While our framework works for general irregular trajectories and select basis gates on them using the approach detailed in Section 2.5, they focus on iSWAP-like (XY) gates with an unwanted CPHASE (XX) component (which belongs to the FSim gate set so is not truly non-standard) and always use CPHASE(ψ)iSWAP($\pi/4$) because it has similar expressivity as iSWAP($\pi/4$) for small deviation ψ. They do not discuss calibration. Their baseline for evaluation is similar to the baseline in our case study, which is to make the trajectory more standard by lengthening the gate duration.

Recent research from both the experimental [42, 86, 132, 57] and theory sides has utilized 2Q (and 3Q) basis gates from a continuous set of standard gates, as opposed to only building and compiling with the best-known gates like CNOT and iSWAP. The works that are most relevant to this project are those that look for a small set of 2Q basis gates (from a continuous standard gate set) that are the most valuable to calibrate. Lao et al. [74] use a numerical approach to test the performance of different gates from the fSim and XY gate sets on a range of application circuits, with the overall circuit success rate as the objective. Peterson el al. [94] from IBM use analytic techniques to find that the gate set $\{CX, CX^{1/2}, CX^{1/3}\}$ is almost as good as the entire continuous set of XX gates in implementing random 2Q gates. They try to minimize the expected (average) infidelity in implementing random 2Q gates under an experimentally motivated error model. Huang et al. [61] proposes using the $\sqrt{iSWAP}$ as 2Q basis gate, instead of using iSWAP or CNOT, and implement it using a 2-fluxonium qubit device. Recent proposals for novel nonstandard 2Q gates in the supercon-

ducting qubit literature that are informed by the current experimental challenges in scaling up with standard 2Q gates include [93, 133].

2.4 Systematic deviations in 2Q gates

The 2Q gate is a critical building block that must be well-engineered before it is used to construct a quantum computer with many qubits. In practice, engineering 2Q gates in the lab involves iterating prototypes of the devices to minimize any and all systematic errors that result from imperfect device design or control along with nonuniformities in device fabrication.

Even if unwanted crosstalk between the qubits is successfully reduced and the 2Q gate is shown to be an effective entangler with a consistent identity, if the gate's identity is somehow nonstandard, one would normally assume it is not useful. The constraint of requiring 2Q gates to be standard is most burdensome for the superconducting qubit platform, where device Hamiltonians are engineered from scratch and there is no 2Q gate that is truly native to the platform - unlike, for instance, the SWAP gates that are native to atomic qubits [124].

Today's multi-qubit superconducting devices are not able to perform perfectly identical 2Q gates between every pair of qubits because of device-level imperfections, tradeoffs and uncertainties. Experimentalists model the expected rate of information leakage between on-chip elements using microwave circuit design software [3, 2], but it is inevitable that irregularities arise during device fabrication and packaging. The devices are at least partially handmade and every fabrication tool has a finite precision. Also, the various materials that make up the layers of the superconducting device can host physical two-level systems that act as sources of noise and even can coherently interact with qubits [105, 112]; reducing the effect of these two-level systems is an active field of research [125]. Another active field of research is reducing irregularities in the fabrication of Josephson junctions, which are critical on-chip elements [68, 54]. For a given device, it can be difficult for the experimentalist to

determine whether a systematic 2Q gate deviation is caused by an imperfection in the device design or in its control. For example, a common source of systematic 2Q gate deviation is the imperfect mitigation of the static ZZ crosstalk which is a dominant source of 2Q gate error for transmon qubits [88, 71, 117, 91, 64, 138]. Devices can be designed to suppress the static ZZ crosstalk but unless the device is properly fabricated, packaged, biased and controlled there will be nonzero static ZZ crosstalk which will cause the 2Q gate to deviate from the target unitary.

Superconducting devices can also have higher order Hamiltonian terms that result in the experimentally measured Cartan trajectory of 2Q gates deviating from the expected Cartan trajectory. This deviation is particularly significant for fast gates enabled by large coupling or large drive strength [96, 84, 63]. Experimentalists have historically tried to suppress these deviations by reducing the 2Q gate drive strength, which has the negative consequence of slowing the 2Q gate down. It is in general difficult to accurately model the effect of the strong drives that perform fast 2Q gates on the Hamiltonian level, and this is an active field of research [96].

Plotting measured 2Q gates in Cartan coordinates is a valuable tool experimentalists can use to easily visualize and study any deviations their gates may have from the expected Cartan trajectory. For example, Figure 2.2 shows a measured Cartan trajectory that is nonstandard. This experimental data was collected from one of the first iterations of a superconducting device [52] that was designed to implement a recently proposed entangling gate architecture [96]. The data includes a very fast (13 ns) perfect entangler. Since the measured trajectory was systematically offset from the predicted one (XY), the experimentalists investigated potential sources of that systematic offset. Since this source of deviation could be eliminated with better device and control engineering, the experimentalists began to optimize their next device iteration accordingly. But in this work we suggest that there is nothing inherently unusable about measured Cartan trajectories that are nonstandard due

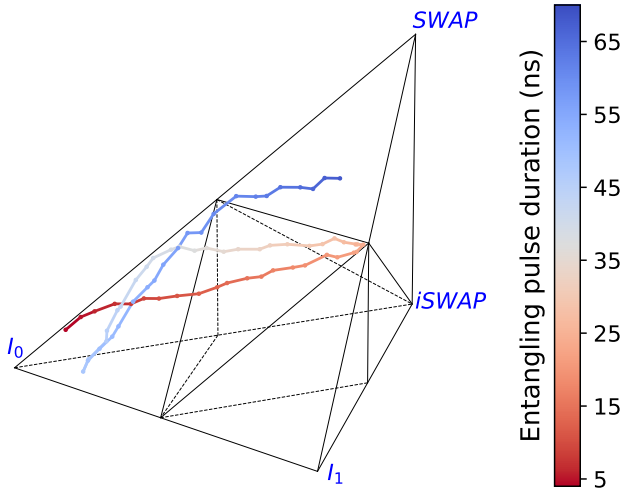


Figure 2.2: Experimental data showing a nonstandard Cartan coordinate trajectory. An experimental implementation [52] of the iSWAP gate with the entangler architecture proposed in [96] yielded a nonstandard Cartan coordinate trajectory close to the plane of I_0, SWAP, and iSWAP. The first instance of a perfect entangler was at an entangler duration of 13 ns. In this nonstandard trajectory, the 13 ns entangler is offset from the Cartan coordinate for the square root of iSWAP and the 26 ns entangler is likewise offset from the Cartan coordinate for iSWAP. Note that due to an experimental hardware constraint the shortest possible entangling pulse duration was 4 ns, so the measured Cartan trajectory begins there.

to this kind of coherent systematic offset, and that the 13 ns nonstandard perfect entangler identified in Figure 2.2 could be treated as a native 2Q basis gate by the compiler.

Our work seeks to enable the use of the nonstandard 2Q gates that can be native to superconducting devices. If 2Q gate calibration and compiling protocols became more flexible, usable superconducting 2Q gate yield would increase considerably, enabling more rapid and effective prototyping of 2Q gates which could be scaled to a computer. Furthermore, any number of novel superconducting devices with very fast 2Q gates that happen to be nonstandard could be effectively utilized for computing.

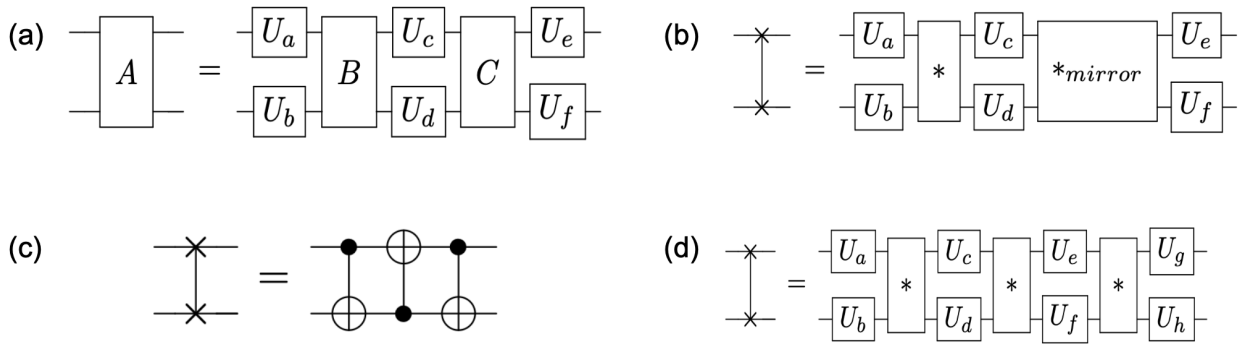


Figure 2.3: (a) Gate A, decomposed into 2 layers with 2Q gates B, C and 1Q gates U_a,U_b,U_c,U_d,U_e, U_f. (b) A general 2-layer decomposition of the SWAP gate. Here $*, *_{mirror}$ can be replaced by any pair of 2Q gates capable of synthesizing a SWAP in 2 layers. (c) The SWAP gate, decomposed into 3 CNOT gates. (d) A general 3-layer decomposition of the SWAP gate. Here the $*$ can be replaced by any 2Q gate capable of synthesizing a SWAP in 3 layers.

2.5 Identifying good 2Q basis gates

2.5.1 Fidelity of a synthesized gate

If a 2Q quantum gate is not directly supported on a device, it needs to be implemented by alternating layers of 1Q and 2Q gates from the set of basis gates that are directly supported. See Figure 2.3 for examples. We say that a decomposition is n-layer if it contains n layers of 2Q gates. Besides the errors that come from noises in the quantum hardware, a synthesized gate also suffers from the approximation error in gate decomposition. Thus the total fidelity of a gate should be the product of the hardware-limited fidelity and the decomposition fidelity. In this work, the decomposition errors are negligible compared to the hardware errors.

In our error model, decoherence is the dominant source of hardware error. So two factors determine whether a 2Q gate set is ideal for synthesizing a target gate: the duration of the basis gates, and the depth of the decomposition circuit. We need to take both into account when deciding on a strategy for selecting basis gates.

2.5.2 An analytic method for determining 2Q circuit depth

When deciding whether a potential basis gate is ideal for synthesizing a target gate, we consider the depth of the decomposition circuit as one of the factors. Given a 2Q target gate A, and a 2Q gate B (or a gate set S), how to determine the minimum circuit depth required for a decomposition of A into B (or S) and 1Q gates? One can take a practical, numerical approach to finding this decomposition. For a given number of layers, one can fix the 2Q gates and then numerically search for the 1Q gates that can minimize the discrepancy between the target unitary and the synthesized gate. One can start the numerical search from 1 layer, and increment the number of layers until the decomposition error gets below a threshold. But a more efficient and accurate way to determine the circuit depth is to apply the analytic method developed by Peterson et al. [95].

Without going into the technical details, here we summarize a key result from [95] that we adapt and apply in Section 2.5.3 and 2.5.4.

Theorem 2.5.1. *There exists a 2-layer decomposition of 2Q gate A into B, C, and 1Q gates as in Figure 2.3(a), if and only if any of the 1 to 8 sets of 72 inequalities that depend on the non-local parts of A, B, C is all satisfied.*

For details of the theorem, the readers can look at Theorem 23 of [95] or the implementation of the function in our code [1]. Note that Reference [95] characterizes the space of 2Q gates with LogSpec instead of the Cartan coordinates (see [95] for the definition of LogSpec). Both are valid ways to represent the non-local part of a 2Q gate, but care must be taken when converting between the two. A gate U usually maps to 1 point in the Weyl chamber, but it usually maps to 2 points in the LogSpec space: $LogSpec(U) = (a, b, c, d)$ and $\rho(LogSpec(U)) = (c + \frac{1}{2}, d + \frac{1}{2}, a - \frac{1}{2}, b - \frac{1}{2})$. If $LogSpec(U) = \rho(LogSpec(U))$ for all A, B, and C, we only need to check one set of inequalities. If $LogSpec(U) \neq \rho(LogSpec(U))$

1. Our code can be found at `https://github.com/SophLin/nonstandard_2qbasis_gates`

for 1, 2, or all 3 of A, B, and C, we need to plug in different versions of the LogSpec and check 2, 4, or 8 versions of the 72 inequalities, respectively.

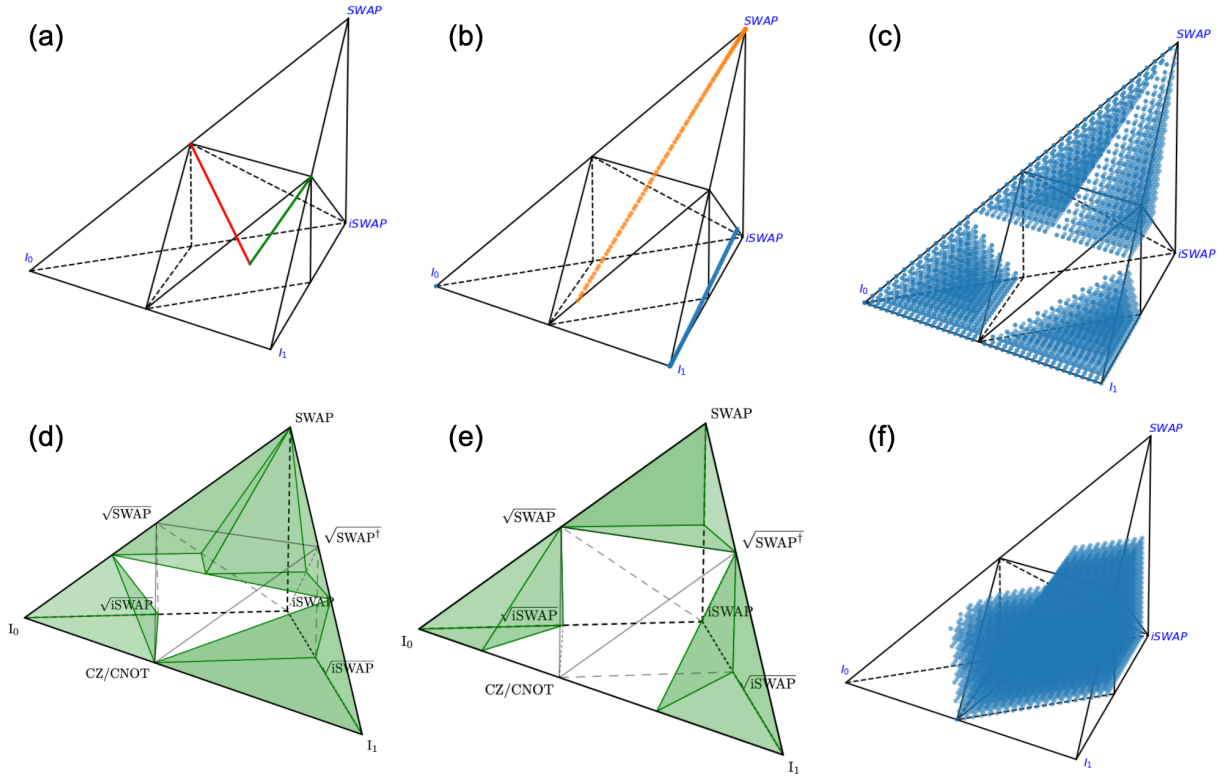


Figure 2.4: (a) Gates that are able to synthesize SWAP in 2 layers form 2 line segments in the Weyl chamber. The red one is from the B gate to $\sqrt{SWAP}$, and the green one is from the B gate to $\sqrt{SWAP}^\dagger$. (b) Pairs of gates that are able to synthesize a SWAP in 2 layers. In blue is an example trajectory that deviates from the standard XY interaction, in orange are the points that would complement the blue ones in synthesizing a SWAP in 2 layers. (c) Gates that are NOT able to synthesize a SWAP in 3 layers. (d) Gates that are NOT able to synthesize a SWAP in 3 layers. The 4 tetrahedra are defined by vertices $\{I_0, CZ, (\frac{1}{4}, \frac{1}{4}, 0), (\frac{1}{6}, \frac{1}{6}, \frac{1}{6})\}$, $\{CZ, I_1, (\frac{3}{4}, \frac{1}{4}, 0), (\frac{5}{6}, \frac{1}{6}, \frac{1}{6})\}$, $\{SWAP, (\frac{1}{2}, \frac{1}{6}, \frac{1}{6}), (\frac{1}{6}, \frac{1}{6}, \frac{1}{6}), (\frac{1}{3}, \frac{1}{3}, \frac{1}{6})\}$, and $\{SWAP, (\frac{1}{2}, \frac{1}{6}, \frac{1}{6}), (\frac{5}{6}, \frac{1}{6}, \frac{1}{6}), (\frac{2}{3}, \frac{1}{3}, \frac{1}{6})\}$. (e) Gates that are NOT able to synthesize CNOT in 2 layers. The 3 tetrahedra in the plot are defined by vertices $\{I_0, (\frac{1}{4}, 0, 0), (\frac{1}{4}, \frac{1}{4}, \frac{1}{4}), \sqrt{SWAP}\}$, $\{I_1, (\frac{3}{4}, 0, 0), (\frac{3}{4}, \frac{1}{4}, 0), \sqrt{SWAP}^\dagger\}$, and $\{SWAP, \sqrt{SWAP}, \sqrt{SWAP}^\dagger, (\frac{1}{2}, \frac{1}{2}, \frac{1}{4})\}$. (f) Gates that are able to decompose SWAP in 3 layers and CNOT in 2 layers.

2.5.3 Synthesis of the SWAP gate

On bounded connectivity architectures, SWAPs make up a significant portion of all two-qubit gates. A SWAP gate exchanges the quantum states of two neighboring qubits. A 2Q gate in a quantum program can be directly scheduled if it acts on two physical qubits that are connected to each other, but this is not the case in general. Superconducting devices are usually designed to have sparse connectivity, because otherwise crosstalk errors would be difficult to suppress. As a result, quantum programs usually contain a large proportion of SWAP gates after they are compiled to run on a superconducting device.

When we select the 2Q basis gate set for each pair of qubits, a top priority is to optimize the fidelity of the SWAP gate that is built from the gate set. We discuss three approaches towards synthesizing a SWAP gate: decompose it into 1, 2, or 3 layers of hardware 2Q gates.

SWAP in 1 layer: This requires a basis gate that is locally equivalent to SWAP. In other words, the trajectory of the available native gates needs to pass through the top vertex of the Weyl chamber.

SWAP in 2 layers: We consider 2 cases: 2-layer decomposition of SWAP using a single 2Q basis gate, and using two different 2Q basis gates.

In the first case, the set of 2Q gates that are capable of synthesizing SWAP in 2 layers are represented by 2 line segments in the Weyl chamber as shown in Figure 2.3(b). One is from the B gate to $\sqrt{SWAP}$ and the other is from B to $\sqrt{SWAP}^\dagger$. We denote them by L_0 and L_1, respectively.

In the second case, for each point $*$ in the Weyl chamber, (as derived in 2.10.2) there is exactly one point $*_{mirror}$ such that they together enable a 2-layer decomposition of SWAP (see Figure 2.3(b)). The line segment from $*$ to $*_{mirror}$ always has one of L_0, L_1 as its perpendicular bisector. Thus, given $*$, we can locate $*_{mirror}$ by rotating $*$ by π around the closer one of L_0, L_1. One example pair of such points is CNOT and iSWAP. For a trajectory that deviates from the standard XY trajectory (goes from Identity to a point near iSWAP),

its "mirror" is a trajectory from SWAP to a point near CNOT (Figure 2.4(b)). Since there's no overlap between the example trajectory and the "mirror", we conclude that the trajectory does not contain any pair of points that is able to synthesize SWAP together in 2 layers.

SWAP in 3 layers: It is a well-known result that 3 invocations of CNOT are required to implement a SWAP [106]. We show the circuit in Figure 2.3(c). In fact, CNOT and iSWAP share the property that they can synthesize any arbitrary 2Q gate in 3 layers but only a 0-volume set of gates (in the Weyl chamber) in 2 layers [95].

For our purpose, we need to know what other gates are capable of decomposing SWAP in 3 layers. We only consider 3-layer decomposition of SWAP using a single 2Q basis gate as in Figure 2.3(d). Let $S_{SWAP,3}$ denote the set of gates that satisfy our requirement. To determine whether a 2Q basis gate G is in $S_{SWAP,3}$, we first locate the corresponding G_{mirror} such that G and G_{mirror} together can provide a 2-layer decomposition of SWAP. Then we apply Theorem 2.5.1 with G_{mirror} as target and G as basis gate to check if there exists a 2-layer decomposition of G_{mirror} into G.

We apply the method above to a sample of points in the Weyl chamber, and obtain the distribution of gates that are able to synthesize SWAP in 3 layers. Since the complement of the set has a simpler shape, here we show a plot of $\overline{S_{SWAP,3}}$, the points that are not able to synthesize SWAP in 3 layers, in Figure 2.4(c). A visual inspection tells us $\overline{S_{SWAP,3}}$ consists of 4 tetrahedra in the Weyl chamber. After locating the vertices of the tetrahedra, we obtain Figure 2.4(d). We also learn that the volume of $S_{SWAP,3}$ is 68.5% the volume of the Weyl chamber.

A 2Q gate trajectory starts from either I_0 (or I_1) and goes out of the bottom left (or the bottom right) tetrahedron in Figure 2.4(d). If the trajectory does not go directly to SWAP, it will enter $S_{SWAP,3}$ after leaving the bottom tetrahedron that it starts from. Thus, the fastest gate on the trajectory that synthesizes SWAP in 3 layers can be found by locating the intersection of the trajectory with the face $\{CZ, (\frac{1}{4}, \frac{1}{4}, 0), (\frac{1}{6}, \frac{1}{6}, \frac{1}{6})\}$ or $\{CZ, (\frac{3}{4}, \frac{1}{4}, 0), (\frac{5}{6}, \frac{1}{6}, \frac{1}{6})\}$.

Summary: Given a 2Q gate trajectory that deviates from XY or XX, the most suitable 2Q gate for SWAP synthesis is the fastest one on the trajectory that is capable of synthesizing SWAP in 3 layers. Although some gates in the Weyl chamber are able to synthesize SWAP in 1 or 2 layers, it is unlikely that the early part of the trajectory overlaps any of them.

2.5.4 Synthesis of other gates

The techniques that we use to study the synthesis of SWAP also applies to other 2Q gates. For example, by applying Theorem 2.5.1 to a sample of points in the Weyl chamber, with CNOT as target, we learn that the gates that are able to synthesize CNOT in 2 layers (denoted $S_{CNOT,2}$ here) takes up 75% of the volume in the Weyl chamber. The complement $\overline{S_{CNOT,2}}$ consists of 3 tetrahedra, as shown in Figure 2.4(e). Therefore, on a 2Q gate trajectory, we can locate the fastest gate that synthesizes CNOT in 2 layers by taking the intersection of the trajectory with the face $\{(\frac{1}{4},0,0),(\frac{1}{4},\frac{1}{4},\frac{1}{4}),\sqrt{SWAP}\}$ or $\{(\frac{3}{4},0,0),(\frac{3}{4},\frac{1}{4},0),\sqrt{SWAP}^\dagger\}$. We can also locate the fastest gate from the trajectory that can both synthesize CNOT in 2 layers and synthesize SWAP in 3 layers, by taking the first intersection of the trajectory with $S_{CNOT,2} \cap S_{SWAP,3}$ (See Figure 2.4(f)).

2.5.5 A strategy for locating good 2Q basis gates

Our framework allows one to prioritize different combinations of target 2Q gates. In Section 2.8, we test the following two criteria for selecting 2Q basis gates from native 2Q trajectories.

1. Select the fastest gate on the trajectory that can synthesize SWAP in 3 layers.

2. Select the fastest gate on the trajectory that can both synthesize SWAP in 3 layers and synthesize CNOT in 2 layers.

As explained in Section 2.5.3, the gate that meets Criterion 1 can be found at the intersection of the 2Q trajectory and one of the 2 faces $\{CZ,(\frac{1}{4},\frac{1}{4},0),(\frac{1}{6},\frac{1}{6},\frac{1}{6})\}$ and $\{CZ,(\frac{3}{4},\frac{1}{4},0),(\frac{5}{6},\frac{1}{6},\frac{1}{6})\}$.

And as explained in Section 2.5.4, the gate that meets Criterion 2 can be found similarly. With this insight, we can locate a desired 2Q basis gate in an experimental setting using the methods in Section 2.6.

Our framework can be easily adapted to other criteria for selecting basis gates. For instance, we can select the fastest gate that can decompose another set of target gates within a certain number of layers. We can also incorporate other metrics like the entangling power into a criterion, e.g. we can locate the fastest gate on the trajectory that is both a PE and can synthesize SWAP in 3 layers.

2.6 Calibration of nonstandard 2Q gates

We propose two stages for calibrating a 2Q basis gate on an unknown trajectory of 2Q gates: first, a more costly "initial tuneup" stage that does not assume any knowledge of the trajectory and then a less costly "retuning" stage that utilizes information from the last initial tuneup and the retunings after it. In a well-controlled industry setup we would imagine the initial tuneup being done once a month and retuning being done daily. In a less well-controlled environment (e.g. one prone to low frequency drift), the initial tuneup could be done more frequently, as needed.

Our proposed calibration approach uses two techniques for experimentally characterizing the unitary of a potentially non-standard 2Q gate: quantum process tomography (QPT) [26, 98] and gate set tomography (GST) [50, 90, 134, 80]. QPT is a simple way to estimate a unitary but it cannot separate state preparation and measurement (SPAM) errors from gate errors [85]. GST is a highly general and accurate tomography technique that characterizes all the operations in a gate set (including SPAM) simultaneously and self-consistently. GST is simple to run, taking minutes to acquire on a superconducting device. GST acquisition is followed by classical processing of the data that can be computed on a cluster in about two hours. Note that during the classical processing, the quantum device can still be used

with gates from the previous calibration cycle. The speedup of GST's classical processing is an active field of research and may be obtained by allowing physics to inform the dominant errors that are expected [4]. The most relevant returns for fine tuning the unitary are the error generators [14] for the gate set. The error generators are a basis for writing the transformation between the measured unitary and the unitary that GST expects. It measures coherent differences and estimates stochastic noise levels. GST is thus a valuable tool for directly characterizing 2Q gates.

Here we list the steps in the initial tuneup stage.

1. Do preliminary coarse tuning experiments such as amplitude and frequency calibration of the entangling pulse drive to estimate the entangling pulse duration of interest. For example, a resonant $iSWAP$-like interaction may have an amplitude and a frequency to tune for optimal population swapping. (5 minutes per pulse)

2. Perform QPT for each 2Q gate in the Cartan trajectory leading up to the approximate 2Q gate of interest. The qubit controller resolution (typically $\sim$1 ns) will determine the spacing between the trajectory points. Based on the findings in Step 1 the trajectory can be cropped around the entangling pulse duration of interest. The unitaries found will be the full list of candidate gates. (30-60 minutes per trajectory)

3. From the candidate gates in the previous steps, use Section 2.5 to identify which of them might be the fastest ones that also are good 2Q basis gates. In this step the list of candidate basis gates is narrowed down. We are not able to narrow down to one basis gate due to the imprecision of QPT.

4. Perform GST to obtain full information about each candidate 2Q gate, including an accurate gate unitary and a breakdown of error sources. Then the set of 2Q basis gates can be chosen. ($\sim$10 minutes for each 2Q gate, followed by classical processing)

The second calibration stage is the quick "retuning" of the 2Q basis gates that relies on the results of the initial tuneup. Once the precise unitary for each 2Q basis gate is found, the gates can be simply retuned using the coarse tuning procedures in Step 1 of the initial tuneup. The information gained in the initial tuneup would allow experimentalists to prescribe a different retuning protocol to each 2Q basis gate according to what it needs. In practice, retuning would most likely be a simple combination of amplitude calibration and frequency calibration of the elements involved in each 2Q basis gate, and it would take approximately 1-5 minutes per 2Q basis gate.

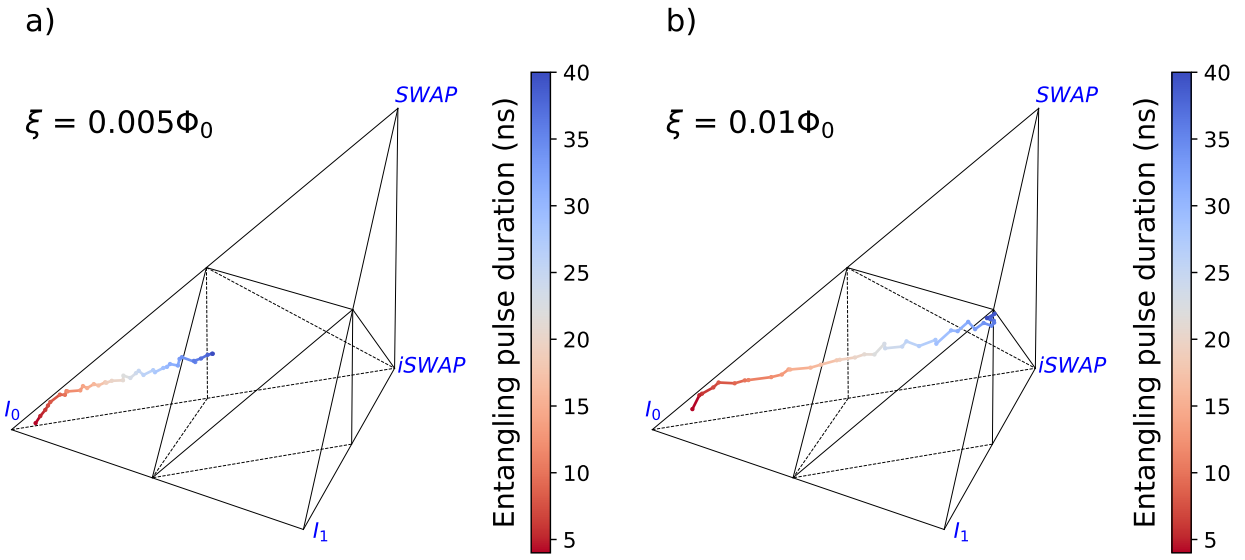


Figure 2.5: Stability over drive amplitude of the experimentally measured Cartan coordinate trajectories. In the same experimental implementation from Figure 2.2, as the entangling pulse drive amplitude ξ increased from $0.005\Phi_0$ to $0.01\Phi_0$, the Cartan coordinate trajectories were found to double in speed but still be qualitatively similar. The data was collected over a two day period. As in Figure 2.2, due to an experimental hardware constraint the shortest possible entangling pulse duration was 4 ns, so the measured Cartan trajectories begin there.

The extent to which previously gathered information can help reduce the cost of retuning depends on the stability of the gate trajectories over time. Figure 2.5 shows the nonstandard Cartan trajectories measured on two days, over two entangling pulse drive amplitudes. Over the five day period that Cartan trajectories were measured for this device, the trajectories were all found to look qualitatively similar, as in Figure 2.5. While limited, this experimental

data suggests that the measured Cartan trajectories obtained in the initial tuneup stage could potentially be used for several days afterward to provide an initial guess for the duration of the good 2Q basis gates.

Our calibration protocol does not include the use of randomized benchmarking (RB) [66, 104, 81]. RB is best suited for architectures with specific target gates that are members of the Clifford group. Furthermore, interleaved RB [82] will estimate the gate infidelity but will provide no information about an error budget. In our setting we do not have a fixed 2Q gate as the goal of implementation and understanding the gate unitaries themselves is a primary goal. We have thus decided GST and QPT are more suitable for precise gate characterization.

The scalability of our proposed calibration method is not different from traditional approaches. Calibration techniques like QPT, RB, and GST can be applied to multiple 2Q gates on the same device in parallel, as long as the gates do not act on the same qubits. One can use an edge-coloring of the device connectivity graph to determine which gates to calibrate simultaneously. An edge-coloring of the grid graph takes 4 colors, one for a sparser connectivity (e.g. heavy hexagonal) takes fewer colors. Thus, for a superconducting device with typical connectivity, the calibration overhead on the quantum device does not scale with the size of the device.

2.7 Compiling with non-standard 2Q basis gates

Most quantum programs and benchmarks are already specified at the 2 or 3 qubit gate level. Therefore, like previous works [61][74][94] that discuss choice of 2Q basis gate and how to use less conventional 2Q basis gates for compilation, we use a transpiler pass to convert other 2Q gates in a circuit into our own 2Q basis gates, instead of building an entirely new compiler.

Some of the prior works decompose 2Q gates from application circuits into 1Q gates and native 2Q gates using a numerical approach [74], while others take an analytical approach

[61] [94]. Note that such a decomposition requires finding the 1Q local unitaries, not just determining the required circuit depth. The analytical and numerical approaches each have their advantages. The numerical approach is more flexible. It can be applied to any 2Q basis and target gates. The analytic methods have limits on what gates they can be applied to, but are faster and some of them guarantee optimal results. There is currently no analytic formula that convert between arbitrary sets of 2Q gates. Huang et al. [61] and Peterson et al. [94] develop analytic algorithms that decomposes an arbitrary 2Q gates into $\sqrt{iSWAP}$ and discrete sets of XX-type gates, respectively. The `decompose_two_qubit_interaction_into_four_fsim_gates` function in Cirq [35] implements an analytic formula that decomposes an arbitrary 2Q gate into 4 layers of a given fSim gate, via the B gate.

In this project, we need to synthesize other 2Q gates from 2Q basis gates that are even less conventional than the ones considered in previous work. Therefore, we take a mostly numerical approach to gate synthesis and write our numerical search code based on NuOp from [74]. The difference is, we use knowledge about decomposition circuit depth computed analytically to inform and speedup the numerical search for 1Q local unitaries. NuOp first attempts to search for a 1-layer decomposition, and moves on to 1 more layer upon failure to find solution, until it meets the target decomposition error rate. Using the analytic techniques for determining circuit depth developed by [95] and extended by our work for SWAP, we are able to skip to the step in NuOp in which a perfect decomposition is guaranteed by theory. This significantly speeds up the numerical search and also guarantee that the solution has optimal depth.

Synthesizing all 2Q gates in the application programs directly into the basis gates might incur a compilation overhead. We avoid it by computing in advance and storing the decompositions of a few common 2Q gates into our basis gates. This only needs to be done once per calibration cycle (usually 1 day) and costs little time. In this work (see Section 2.8) we only directly decompose SWAP and CNOT into our basis gates. But instead of taking

this minimalist approach, one can alternatively prepare decompositions for a larger set of potential target gates into the basis gates. The cost would still quite small. We imagine that one can identify a set of potentially useful target gates using an approach similar to [74], except that [74] looks for a set of gates to calibrate instead of decompose. In addition, in the scenario where programs wait in long queues before execution, one might be able to afford directly decomposing all 2Q gates in the circuits into the basis gates.

2.8 Case study: entangling fixed frequency far-detuned transmons with a tunable coupler

2.8.1 Introduction to the case study entangling gate architecture

Many efforts are being made in industry and academia to design a 2Q entangling gate architecture that can be used for scaling up to a general quantum computer [128, 42, 117]. The all-microwave cross-resonance gate was recently used by IBM to do a high fidelity CNOT gate in 90 ns [128], but to suppress the always-on ZZ crosstalk mentioned in Section 2.4, precise crosstalk cancellation pulses applied to both qubits during run time were required, adding complexity to the architecture. Google Quantum AI and MIT have each developed entangling gate architectures for high fidelity CZ and iSWAP gates, with Google's architecture supporting a continuous set of these standard gates [42, 117]. Google's architecture requires all qubits and couplers to be flux-tunable, which adds complexity and additional sources of leakage and noise to their architecture. Similarly, in order to suppress the always-on ZZ crosstalk, MIT's architecture requires one qubit per pair to be tunable as well as the coupler.

The unit cell of our case study entangling gate architecture is a pair of qubits and a coupler. This unit cell, first proposed in [96], was designed to perform a diverse set of 2Q gates, including iSWAP and CZ; the full list of 2Q gates can be found in Table 1 of [96]. The

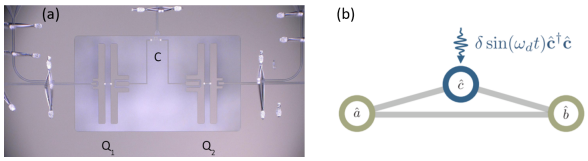


Figure 2.6: (a) Optical image of the device presented in [52] shows two fixed frequency transmons coupled via a tunable coupler. (b) Schematic for modelling the device adapted from [96].

two qubits are fixed frequency transmon qubits; the benefits of fixed frequency transmons are that they are easy to fabricate and can be reliably engineered to have high coherence $>$ 100 us [97]. The two qubits are also far detuned from each other so there is reduced single qubit control crosstalk. The coupler is a generalized flux qubit which has been designed to have several good properties for qubit control. Notably, because the coupler's positive anharmonicity has been designed to balance out the negative anharmonicity of the two qubits, the eigenspectrum of this architecture's unit cell can support a zero-ZZ crosstalk bias point. This architecture is relatively simple to implement because fixed frequency transmons have high coherence, there is only one flux-tunable element in the unit cell (the coupler), and it is easy to bias the unit cell to zero-ZZ crosstalk.

A model Hamiltonian of the two qubits coupled with a tunable coupler is shown in Section 2.10.1. Here we highlight the time-dependent term, $\hat{H}_c(t)$ that describes the coupler dynamics:

$$\hat{H}_c(t) = \omega_c(t)\hat{c}^\dagger\hat{c} + \frac{\alpha_c}{2}\hat{c}^{\dagger 2}\hat{c}^2 \tag{2.2}$$

where α_c is the coupler anharmonicity, $\hat{c}$ is the annihilation operator and the coupler frequency $\omega_c(t)$, corresponding to the transition to its first excited state, can be varied in time via the flux through its superconducting loop. Low-crosstalk 2Q gates are realized by AC modulating this coupler frequency after DC biasing it to the zero-ZZ crosstalk bias point.

In [52] an early prototype device (shown in Fig. 2.6) for this case study architecture demonstrated a fast perfect entangler biased to zero-ZZ crosstalk. This device produced

the nonstandard 2Q gate trajectory shown in Figure 2.2, which included a 13 ns perfect entangler. Figure 2.5 shows how the measured trajectories were similar over a range of entangling pulse drive amplitudes that did not exceed $\xi = 0.01\Phi_0$, the point at which strong drive effects would be expected to become non-negligible [96]. So in this early prototype device, the measured trajectories in Figures 2.2 and 2.5 were not nonstandard because of strong drive effects, but because of some other systematic in the experiment.

2.8.2 Our simulation approach

The case study entangling gate architecture natively supports strong parametrically activated interactions between the two qubits. Since the full Hamiltonian for this architecture is computationally intensive to model [96], for our simulation we use the simplified effective Hamiltonian from [96] that models the device using fewer parameters while still capturing all of the essential physics of the device (see 2.10.1). Our general protocol for simulating Cartan trajectories is as follows:

1. We input the simulated device parameters into our Hamiltonian. These parameters include the qubit frequencies ω_a and ω_b, and the qubit coherence times. This generates the eigenspectrum of the simulated device.

2. We bias the coupler frequency (ω_c^0) between the two qubit frequencies (ω_a, ω_b) such that the static ZZ term (i.e. for $\delta(t) = 0$) between the two qubits is tuned to zero.

3. We specify the drive amplitude ξ of our entangling pulse. In this case study we implement a iSWAP-like entangler, so the entangling pulse is driven at the frequency ω_d that generates maximal population swapping between the two qubits. For $\xi \leq 0.01\Phi_0$, the entangling pulse frequency ω_d is essentially identical to the difference frequency of the two qubits $|\omega_a - \omega_b|$. However, increasing $\xi > 0.01\Phi_0$ activates the two-photon process in Equation 2.2, causing population to enter the second excited state of the

coupler and modify the entangling interaction. This in turn causes ω_d to deviate from $|\omega_a - \omega_b|$. The entangling pulse is modulated by a rectangular envelope, as was done in experiment to obtain the measured trajectories; due to qubit controllers typically having a time resolution of 1 ns, short entangling gates $\sim$10 ns have to be implemented using a pulse with a fast rise time. Experimentalists typically choose between a flat top Gaussian pulse with a short rise time, or a rectangular pulse for simplicity.

4. We evolve the time-dependent Hamiltonian and project the evolution propagator on the computational subspace to obtain the effective unitary operation with respect to the entangling pulse drive duration. This time ordered sequence of unitary operations can be represented as a trajectory in the Weyl space using Cartan coordinates. By examining the trace of the effective unitary propagator we can obtain the leakage outside the computational space. We confirm that the leakage rates are much below the expected gate errors due to decoherence.

In this case study we simulate standard and nonstandard 2Q trajectories. The simplest and most consistent way to do this is to use the same simulated devices but to vary the drive power ξ. For $\xi \leq 0.01\Phi_0$ we expect the above protocol to result in a standard iSWAP interaction between the two qubits. But for $\xi > 0.01\Phi_0$, we expect strong drive effects to begin to emerge and cause the Cartan trajectory to deviate away from a standard iSWAP. We note that the simulated trajectories differ in several ways from the measured trajectories in Figures 2.2 and 2.5. Firstly, the measured trajectories are nonstandard even for $\xi \leq 0.01\Phi_0$ due to an additional systematic effect in the experiment. Secondly, the simulated trajectories are consistently slower than the measured trajectories; e.g. at $\xi = 0.01\Phi_0$, the simulated trajectories are slower by a factor of 3.5 than the measured trajectory, which included a 13 ns $\sqrt{iSWAP}$-like entangling gate. These discrepancies can both be explained by the simulation model Hamiltonian being significantly simpler than the true device Hamiltonian. Aside from these discrepancies, our simulations are realistic; our trajectories are generated

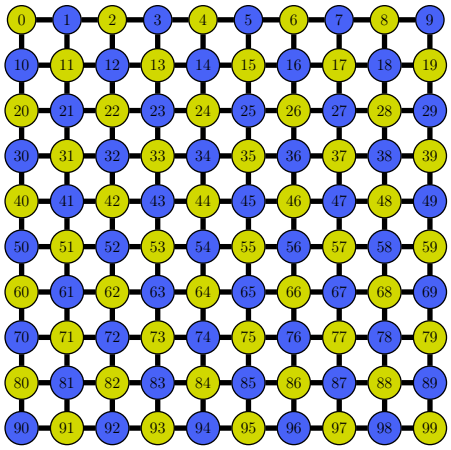


Figure 2.7: Device simulation. The high and low frequency qubits are shown in different colors. Each edge connects two qubits with different colors.

using parameters and techniques that closely resemble those used in experiment and our method for generating standard and nonstandard trajectories using a single simulated device is physically intuitive.

Simulating Cartan trajectories over a range of entangling pulse amplitudes ξ we observe the correct intuitive behavior. The simulated trajectories deviate more and more from the standard iSWAP as the entangling pulse amplitude increases beyond $\xi = 0.01\Phi_0$. Secondly, the speed of the simulated trajectories scales linearly with ξ. This agrees with the experimental data shown in Figure 2.5 where the measured trajectory doubled in speed when ξ increased by a factor of two.

2.8.3 Methodology

We simulate a 10 by 10 device with grid connectivity (Fig. 2.7),where the qubit frequencies of each pair of neighbors are sampled from two normal distributions respectively with means that differ by 2 GHz. We use a 5% standard deviation for sampling the qubit frequencies. Improved fabrication techniques have reduced the smaller standard deviation to about 0.5% [54], but we use a larger standard deviation to show that our method is robust to variations in device fabrication.

Between each pair of neighboring qubits on the 10×10 grid, we simulate two types of 2Q trajectory by varying the entangling pulse amplitude ξ: 1) A baseline trajectory generated with a low entangling pulse amplitude of $\xi = 0.005\Phi_0$ and 2) a nonstandard trajectory due to strong drive effects resulting from a larger $\xi = 0.04\Phi_0$.

Then on each nonstandard trajectory, we select 2Q basis gates using Criterion 1 and 2 (respectively) introduced in Section 2.5.5. We test these 3 sets of 2Q basis gates on common application circuits as benchmarks. We use the Qiskit[8] transpiler with the "SABRE"[75] layout and routing methods to map the benchmarks circuits to the 10×10 grid connectivity. With the nonstandard basis gates, we compile circuits using the methods from Section 2.7. With the $\sqrt{iSWAP}$ from the standard trajectories, we use the analytic approach in [61]. Like the 2Q basis gates selected with Criterion 2, $\sqrt{iSWAP}$ decomposes SWAP in 3 layers and CNOT in 2 layers, but we can also use it to directly decompose other 2Q gates (like the CRZ gates in the QFT benchmarks) analytically. For the 1Q gates in the gate and circuit synthesis, we use a duration of 20 ns, which is typical for fixed-frequency transmon qubit processors [63].

Decoherence is the dominant hardware noise in our noise model, because crosstalk is suppressed by the high detuning in the qubits. For each qubit, we model the decoherence error as $1 - e^{-t/T}$, where T is the coherence time of the qubit. We set T to a typical value of 80 μs for all qubits. We compute t as $t_f - t_i$, where t_i is the start of the first gate on the qubit and t_f is the end of the last gate on the qubit. The total coherence-limited fidelity of a circuit is the product over the $e^{-t/T}$ term from each qubit. The decomposition errors in gate synthesis are negligible compared to the decoherence errors, and can be reduced to arbitrarily close to zero in theory. Thus we only show the coherence-limited fidelities in the results.

2.8.4 Results

Before discussing our results, as a disclaimer we note that while increasing the entangling pulse drive amplitude is one way to speed up 2Q gates, it is by no means an all-purpose solution that we generally advocate for. We chose to do this in our simulation case study only because it was a simple and intuitive way to compare standard and nonstandard simulated gates for the same case study entangling architecture. For this case study architecture, the drive amplitudes chosen were realistic in an experimental setting.

	Basis	SWAP	CNOT
Baseline	83.04 ns 99.884%	329.1 ns 99.541%	226.1 ns 99.684%
Criterion 1	10.15 ns 99.986%	110.5 ns 99.845%	110.5 ns 99.845%
Criterion 2	10.76 ns 99.985%	112.3 ns 99.843%	81.51 ns 99.886%

Table 2.1: Average duration (top) and coherence-limited gate fidelity (bottom) of the 2Q basis gates and the synthesized SWAP and CNOT gates, from baseline, Criterion 1, and Criterion 2.

The average durations and coherence limited fidelities (obtained using the Qiskit Ignis `coherence_limit` function [1]) of the synthesized SWAP and CNOT gates from the two approaches are summarized in Table 2.1. In Table 2.2, we show the coherence-limited circuit fidelities of 5 sets of benchmark circuits, when transpiled to different sets of 2Q basis gates.

We first observe that the faster nonstandard 2Q basis gates have $\sim$8x lower coherence-limited infidelities than the baseline standard 2Q gates. We also observe that the synthesized SWAP (CNOT) gates from Criterion 1 and 2 are 3.0x and 2.9x (2.0x and 2.8x) faster than the baseline, respectively. Due to the relation between gate fidelity and circuit fidelity, fidelity improvements scale exponentially in benchmark size.

Next, we observe that Criterion 2 performs better than Criterion 1. This is not surprising since it has significantly faster CNOT gates and only slightly slower SWAP gates compared

to Criterion 1.

For the baseline case, the 1Q gate duration is 4x shorter than the standard 2Q basis gate, and therefore ∼24% of the duration of the compiled SWAP/CNOT gate is spent performing 1Q gates. In contrast, for the nonstandard case, the 1Q gate duration is 2x longer than the nonstandard 2Q basis gate, and ∼72% of the duration of the compiled SWAP/CNOT gate is spent performing 1Q gates. This puts us in the regime of today's fastest large superconducting processors such as Google's Sycamore device, where the optimal processor configuration that minimizes the overall effects of gate error has the 1Q gates being roughly twice as long as the 2Q gates [9].

Benchmark	Baseline	Criterion 1	Criterion 2
qft 10	58.2%	65.6%	70.8%
qft 20	1.33%	6.03%	9.94%
bv 9	88.7%	94.4%	95.3%
bv 19	79.3%	89.9%	91.0%
bv 29	44.5%	72.5%	74.3%
bv 39	26.8%	56.3%	59.7%
bv 49	27.7%	58.4%	62.4%
bv 59	12.5%	43.8%	47.4%
bv 69	9.15%	39.4%	43.2%
bv 79	0.428%	11.3%	14.2%
bv 89	2.44%	23.1%	26.3%
bv 99	0.06%	6.26%	7.97%
cuccaro 10	21.5%	46.3%	52.6%
cuccaro 20	0.800%	7.68%	11.8%
qaoa_0.1 10	97.2%	98.5%	98.8%
qaoa_0.1 20	84.4%	92.0%	93.6%
qaoa_0.1 30	14.4%	43.3%	49.0%
qaoa_0.1 40	0.00585%	5.59%	8.56%
qaoa_0.33 10	66.1%	81.0%	84.3%
qaoa_0.33 20	15.0%	42.2%	48.2%

Table 2.2: The decoherence-limited fidelities of benchmark circuits, transpiled using the standard 2Q basis gates from baseline, and the nonstandard ones selected by Criterion 1 and 2. The QAOA benchmarks all have $p = 1$ where p is the number of times the protocol is repeated. The fractions 0.1 and 0.33 are the probablities that an edge is created between a pair of nodes.

2.9 Conclusion

The idea of a uniform set of basis gates naturally arose from early notions of universal gate sets, which experimentalists then implemented on various qubit platforms. By looking at the theory of possible entanglers, we have found that there are many options for good 2Q basis gates, and that these gates behave differently on each pair of interacting qubits in a processor. This led us to a radically new idea, why be constrained to a single canonical gate (e.g. CX or CZ)? Why not tune up the gate that will have the highest fidelity between every pair of qubits, allowing each to differ and instead adjust for these variations in software? If we do not treat all the coherent deviations in gate trajectories as errors, we will have more freedom in hardware design and achieve a higher gate fidelity.

In this paper, we examined the space of possible entanglers and developed a practical method for finding a high-fidelity entangler between every pair of qubits. In the case study, we find heterogeneous basis gates that are ∼8x faster than the baseline, and use them to synthesize faster SWAP and CNOT gates than synthesized by the baseline $\sqrt{iSWAP}$ gate from the standard XY-type trajectories. We then evaluate these heterogeneous basis gates on a number of benchmark circuits and find fidelity improvements that scale exponentially in benchmark size.

Our approach successfully uses software to overcome the limitations of today's hardware. Such types of adaptive basis-gate design will be essential to pioneering innovative future quantum systems.

2.10 Appendices

2.10.1 Hamiltonian of 2 qubits coupled with a tunable coupler

The system Hamiltonian of the two qubits coupled with a tunable coupler can be modelled as in [96]:

$$\hat{H}(t) = \hat{H}_a + \hat{H}_b + \hat{H}_c(t) + \hat{H}_g, \tag{2.3}$$

with

$$\begin{aligned}
\hat{H}_a &= \omega_a \hat{a}^\dagger \hat{a} + \frac{\alpha_a}{2} \hat{a}^{\dagger 2} \hat{a}^2, \\
\hat{H}_b &= \omega_b \hat{b}^\dagger \hat{b} + \frac{\alpha_b}{2} \hat{b}^{\dagger 2} \hat{b}^2, \\
\hat{H}_c(t) &= \omega_c(t) \hat{c}^\dagger \hat{c} + \frac{\alpha_c}{2} \hat{c}^{\dagger 2} \hat{c}^2. \\
\hat{H}_g &= -g_{ab} \hat{a}^\dagger \hat{b} - g_{bc} \hat{b}^\dagger \hat{c} - g_{ca} \hat{c}^\dagger \hat{a} \\
&\quad - g_{ab}^* \hat{a} \hat{b}^\dagger - g_{bc}^* \hat{b} \hat{c}^\dagger - g_{ca}^* \hat{c} \hat{a}^\dagger
\end{aligned} \tag{2.4}$$

where $\omega_{a(b)}$ corresponds to the qubit a(b) frequency, g_{ij} represents capacitive coupling strength between elements i and j. The entangling interaction is realized by modulating the coupler frequency as $\omega_c(t) = \omega_c^0 + \delta \sin(\omega_d t)$.

2.10.2 SWAP synthesis in 2 layers

See the circuit in Fig. 2.3(a). Let $A = SWAP$ we get the equation

$$SWAP = (e \otimes f) C (c \otimes d) B (a \otimes b).$$

Move $e \otimes f$ and $a \otimes b$ to the other side and move $e \otimes f$ through SWAP,

$$\begin{aligned}
C(c \otimes d)B &= (e \otimes f)^\dagger SWAP (a \otimes b)^\dagger \\
&= SWAP (f \otimes e)^\dagger (a \otimes b)^\dagger \\
&= SWAP (fa \otimes eb)^\dagger.
\end{aligned}$$

Move $(fa \otimes eb)^\dagger$ to the LHS, and C to the RHS,

$$(c \otimes d)B(fa \otimes eb) = C^\dagger SWAP.$$

This equation tells us that, B and C can synthesize SWAP as in Fig. 2.3(a) if and only if the Cartan coordinates of B are equal to the Cartan coordinates of $C^\dagger SWAP$ up to canonicalization. Let $B \sim (x, y, z)$ and $C \sim (x', y', z')$, then we have $(x, y, z) \sim (-x', -y', -z') + (\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$. From this we can tell that for every local equivalence class $[B]$ of 2Q gates, there is exactly one local equivalence class $[C]$ such that $[B]$ and $[C]$ together can synthesize SWAP in 2 layers. And since we know how to canonicalize Cartan coordinates into points within the Weyl chamber, given $[B]$ we will be able to find the corresponding $[C]$. Here we do not elaborate on how we identify the geometric relation between $[B]$ and $[C]$ inside the Weyl chamber, but the readers can check our claim by applying Theorem 2.5.1.

CHAPTER 3

CODESIGN OF QUANTUM ERROR-CORRECTING CODES AND MODULAR CHIPLETS IN THE PRESENCE OF DEFECTS

3.1 Introduction

Fault-tolerant (FT) quantum computers will enable the implementation of large-scale quantum algorithms such as search [107] and factoring [51]. These machines are designed to protect quantum information by encoding logical qubits in quantum error-correcting codes that consist of a large number of interacting physical qubits.

Achieving fault tolerance requires a practical way to manufacture physical qubits. One leading approach is to manufacture a large planar array [41] of qubits in a solid-state system such as superconducting [70, 116, 7] or spin qubits [43, 53]. In these hardware architectures, Ref. [92] estimates that over a million high-fidelity qubits will be required for FT quantum applications. However, scaling a quantum device to this magnitude presents significant challenges.

The fabrication of large-scale solid-state devices encounters additional complexities due to fabrication errors. There are many steps in processing where a slight deviation from the target specification occurs due to process imprecision or stochastically appearing impurities or imperfection[55, 67]. As a result, variation in the quality of qubits, as well as the links along which qubits interact, is inevitable. On quantum devices, physical qubits will exhibit inhomogeneous characteristics, including variations in gate success, measurement fidelity, and coherence, and will likely contain *faulty (defective) qubits* — qubits with severely limited functionality. With today's technology, [110] estimated that $\sim 2\%$ of the qubits on a transmon device would be faulty.

To effectively scale up solid-state quantum devices in the presence of fabrication errors, we combine two approaches: (1) Leveraging the flexibility of a modular architecture, which offers

greater adaptability compared to a monolithic architecture by enabling the post-selection of individual chiplets prior to their integration into a larger device. (2) Adapting QEC codes to defective qubit arrays, which enables the utilization of some defective chiplets. The combination of these two strategies helps mitigate the additional resource overhead caused by fabrication errors, but it requires a thorough understanding of the performance of an adapted surface code, and how the overhead is affected by design choices. Importantly, establishing an informed post-selection criterion that accounts for how each chiplet is affected by its defects requires identifying performance indicators specific to the adapted surface code. This also helps us understand how the logical fidelity of an adapted surface code compares with that of a standard surface code.

An adapted surface code can be implemented on a defective chiplet by employing "super-stabilizer measurements," which are capable of detecting errors in proximity to defective areas, as outlined in previous studies [114, 113, 10, 115]. We illustrate the construction of super-stabilizers in Fig. 3.1. Conveniently, these super-stabilizers can be measured in a timely manner using the operational qubits adjacent to the defective region [114, 113, 10]. It has been shown, in theory, that this approach can scale on a large lattice [115]. However, there is a noticeable gap in existing research regarding the comparative performance and resource overhead of such adapted codes relative to the standard, defect-free surface code. To address this, we conduct numerical simulations. And to facilitate our numerical simulations, we develop an automated method to map surface code to defective grids by deforming boundaries and forming super-stabilizers. Notably, our automated method can define a surface code for an arbitrary configuration of defects.

Using our numerical results, we identify two key figures of merit that characterize the fidelity of a surface code on a defective chiplet. The first is the distance on the defective patch d, the least number of physical errors that can lead to a logical failure. In the regime of low error rate per gate, p, we find that to leading order, the logical failure rate decays like

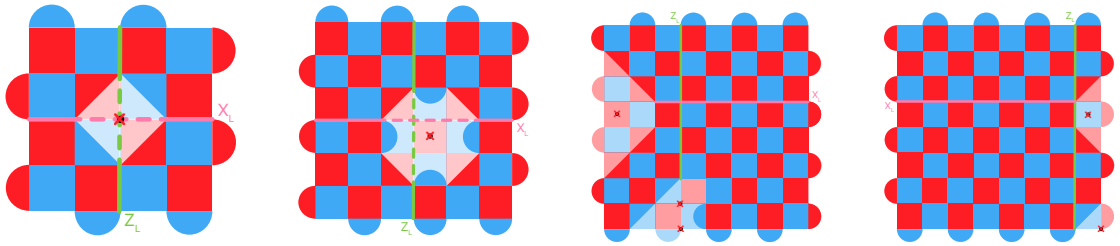


Figure 3.1: Examples of super-stabilizers and boundary deformations. The faulty qubits are marked with a red 'x', and the excluded part in a patch is represented by lighter color. (a) One broken data qubit in the interior, handled by a super-stabilizer. (b) One broken syndrome qubit in the interior, handled by larger super-stabilizers. (c)(d) Broken qubits near the boundary require boundary deformations.

$O(p^{d/2})$, just as for defect-free chiplets. Remarkably, we identify this scaling when physical errors occur at a rate $p \sim 10^{-3}$, the regime where we expect a defect-free chip to be operable. This implies that the defective chips are functionally similar to the defect-free ones with the same d, except that they cost more physical qubits. Aside from this scaling, we identify variation in surface codes with equivalent code distances. We find a second figure of merit that differentiates among these codes. Specifically, we find that the logical failure rate will scale with the number of different ways that a logical failure can occur with d physical errors. Both of these figures of merit can be efficiently computed after the surface code is adapted to a defective grid.

These two indicators enable us to rapidly assess the quality of individual defective chiplets. This is necessary for establishing a post-selection criterion for the modular chiplet architecture, and also for resource overhead evaluation. Our numerical results demonstrate that our post-selection criterion is more effective than the natural strategy to select the chiplets with the fewest defects.

We quantify the resource overhead by the total number of fabricated physical qubits per logical qubit, including the qubits on the chiplets that are not selected. In Fig.3.12b,3.13b, and 3.17b, we show the factor of resource overhead relative to the ideal no-defect case. Our

results show that it is important to select the chiplet size based on the fabrication error rate, in order to achieve a balance between having a high yield (proportion of chiplets that meet the standard) and using a small number of physical qubits per patch of code. When the optimal chiplet size is chosen based on the results, the resource overhead is below 3X and 6X respectively for the two defect models we study (defective links only v.s. same rate of defect on links and qubits), when the defect rate is below 1%. This holds for a wide range of performance targets. Without the ability to tolerate defects, the resource overhead grows exponentially with the number of physical qubits in a logical qubit (which increases for higher performance targets) and also grows faster with the defect rate.

We also evaluate the sensitivity of the yield to two design choices: what boundary constraint is imposed on each patch, and whether the chiplet design allows the data and syndrome qubits to be swapped by rotating a chiplet. Although in this paper we mainly evaluate each patch by its capacity to store a logical qubit, a boundary constraint can ensure the quality of multi-qubit logical gates.

Finally, we identify cutoff fidelity values for determining whether a qubit should be treated as faulty or kept in the code. This is important in the practical setting because the impact of fabrication errors varies and there's not always a clear line between faulty and good qubits.

Summary of contributions and results:

- We develop an automated method that adapts the rotated surface code to a grid with an arbitrary distribution of defects using super-stabilizers. Our automated method produces a simulation of active error correction that is implemented on Stim [45]. Our code and a demo notebook is available at `https://github.com/SophLin/superstabilizer_demo`.

- Using our numerical simulations, we identify effective indicators for assessing the fidelity of a surface code adapted to defective chiplets.

- We present the first evaluation of the impact of fabrication defects on the resource

overhead of quantum error correction. Our focus is on the modular chiplet architecture where one logical qubit is allocated on each chiplet. We quantify the resource overhead as the average number of physical qubits fabricated for a logical qubit, and evaluate its sensitivity to system parameters. We show that the post-selection of chiplets and the ability to use defective chips are both critical for reducing the extra overhead caused by fabrication errors.

- We identify cutoff fidelity values for determining whether a qubit with worse performance than its neighbors should be treated as faulty or kept in the code.

3.2 Background

3.2.1 Fabrication errors and variations on transmon-based quantum devices

In this section, we discuss some of the sources of fabrication errors and variation for transmon qubits. Although our discussion is not exhaustive, it is meant to give some intuition for why current chips see a 2% defect rate.

Imprecision in the fabrication of Josephson Junctions (JJs) is a source of varied qubit performance. A JJ, two superconductors separated by a thin metal-oxide insulator, is the heart of a transmon qubit [44]. JJs have incredibly small feature sizes that are hundreds of nanometers in scale [55], smaller than the wavelengths used during optical lithography. Thus, slight imperfections that appear in JJ positioning, component dimension, or surrounding layers influence operational characteristics of the transmon [67].

On fixed-frequency transmons with fixed couplers, one of the most commonly used superconducting qubits, frequency collision is a dominant type of fabrication error. Fabrication variation can deviate a qubit's frequency from the ideal frequency, resulting in spectral overlaps that cause frequency collisions. This variation is stochastic, causing the resulting frequency profile of each chip to be unique.

Another type of unintended defect that frequently and stochastically appears across a quantum chip during processing is a two-level system (TLS) [87]. A TLS is caused by impurities inside materials or irregularities within atomic crystalline lattice structures appearing unexpectedly in oxide layers or on the surface of the chip. Because of the layered approach associated with transmon processing, there are many opportunities for TLSs to appear during fabrication.

3.2.2 Surface code

The surface code [33, 41, 65] is one of the most practical quantum error-correcting codes for physical realization due to its implementation using a two-dimensional nearest-neighbor qubit layout, and its high tolerance to noise. It can perform a universal set of logic gates while maintaining its local planar layout. One can use lattice surgery to perform entangling operations [59, 22, 76] and magic state distillation [17, 48] to perform non-Clifford gates. For most of this work, we will concentrate on the performance of the surface code storing a logical qubit over time.

In this paper, we use the rotated planar surface code due to its low qubit overhead [16, 40, 13]. We define the surface code with code distance d on a $d \times d$ grid of data qubits. The errors on the data qubits are detected using d^2-1 measurement qubits, otherwise known as ancilla qubits or syndrome qubits. More specifically, a measurement qubit is placed on either a red or a blue face of the grid of data qubits, as in Fig. 3.2. Note also that the faces at the lattice boundary each touch two data qubits.

We measure stabilizers to detect errors that qubits experience. Stabilizers are measured repeatedly in cycles to determine the locations of errors that occur over time. In each cycle, each measurement qubit is used to measure either a Pauli-X stabilizer, $\hat{X}_a\hat{X}_b\hat{X}_c\hat{X}_d$, or a Pauli-Z stabilizer $\hat{Z}_a\hat{Z}_b\hat{Z}_c\hat{Z}_d$, depending on the color of the lattice face on which the measurement qubit lies. The measurement qubits on the boundary only acts on two data

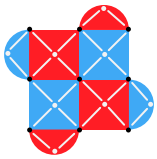


Figure 3.2: Rotated surface code with d=3. The black dots represent the data qubits, and each red (blue) face represents an X (Z) stabilizer. Each stabilizer requires an ancilla qubit.

qubits, and therefore measure stabilizers of the type $\hat{X}_a\hat{X}_b$ or $\hat{Z}_a\hat{Z}_b$. Stabilizers are measured with circuits that are detailed in, e.g., Ref. [121].

In practice, to deal with the errors that occur on data qubits, as well as errors that occur on measurement qubits that may cause stabilizer circuits to give unreliable outcomes, we compare a stabilizer reading at cycle t to the reading of the same stabilizer made at cycle $t-1$. A difference in reading gives rise to an error detection event.

By performing multiple cycles, we obtain a history of detection events over time that we call the error syndrome. In general, we can regard errors as string-like objects in this error syndrome, where detection events occur at the end-points of these strings. See e.g. Refs. [21, 33, 41, 126] for details. Using the error syndrome, we can obtain a correction to recover encoded information using minimum-weight perfect-matching algorithm [21, 33, 41, 56, 126], where we deal with detection events due to Pauli-X stabilizers and Pauli-Z stabilizers separately. We concentrate on only Pauli-Z stabilizers throughout this work, but note that an equivalent analysis will hold for the alternative stabilizers.

A logical error is introduced into the surface code when at least $d/2$ errors occur along a non-trivial path over the surface code error syndrome history [13, 33, 41, 126]. If we assume that an individual error occurs with probability $O(p)$, then in the limit that p is small, we can fit the logical error rate to the ansatz

$$LER = \beta(Np)^{\alpha d}, \tag{3.1}$$

where N and $\alpha \leq 1/2$ are constants to be determined [13, 18, 33, 41, 126, 127].

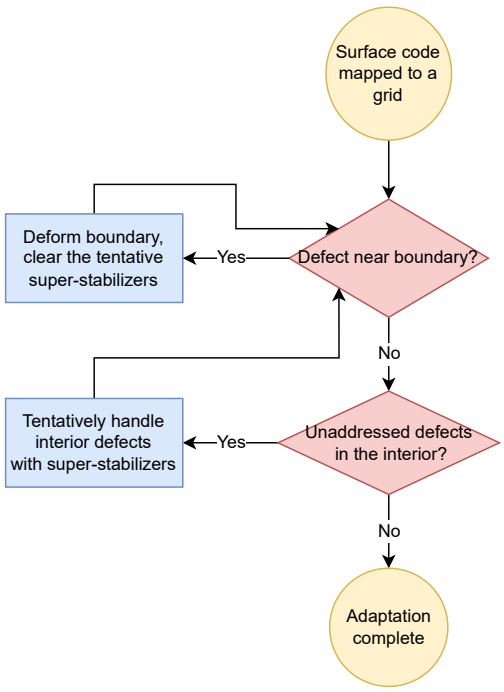


Figure 3.3: Algorithm for mapping rotated surface code to a defective grid by deforming boundaries and forming super-stabilizers.

3.3 Deforming boundary and forming super-stabilizers

Faulty qubits and links can be particularly harmful to the implementation of topological QEC codes if they are not handled correctly. This is because qubit errors in the vicinity of fabrication defects may not be detected. Given a finite density of fabrication defects, this will inhibit the decay of logical failure rate as we increase code distance, if we do not deal with these defects correctly.

Building upon the theory in Ref. [115], we develop and implement an automated method that adjusts a surface code for arbitrary defect distributions. Our algorithm includes deforming boundaries of a code and forming super-stabilizers in the interior. A flowchart is shown in Fig. 3.3. Our code takes the chiplet size l and a list of defects as input, then adapts a surface code to the defective grid and generates a stabilizer measurement circuit compatible with the Stim[45] simulator.

We learn the occurrence of errors close to fabrication defects by forming super-stabilizers

around them. The values of these super-stabilizers can be inferred from local measurements around the defects [114, 113, 10]. Furthermore, we repeat the local measurements used to obtain the super-stabilizer for an elongated time, over a timescale that is commensurate with the size of the super-stabilizer that are defective [115]. A measurement schedule with this feature is adopted in Ref. [115] to demonstrate the procedure helps achieve a threshold — when the physical errors on the non-faulty qubits is lower than the threshold, one can expect the quality of the logical qubit to improve as it is encoded with more physical qubits.

As an example, we show how we can construct a weight-6 X and Z super-stabilizer around a single faulty qubit in the interior of the code in Fig. 3.1a). The value of the X (Z) super-stabilizer can be computed from the direct product of the two weight-3 broken X (Z) stabilizers used as gauge operators. The X and Z gauges anti-commute, so they cannot be measured in the same cycle. Instead, they are measured in alternate cycles.

Another important example is the case where a measurement qubit in the interior of the qubit grid is faulty (Fig. 3.1b). All of its neighboring data qubits are disabled and larger super-stabilizers (each consisting of 4 gauge operators) are formed. Around a larger defect cluster like this, instead of measuring the X and Z gauges in alternate cycles (XZ...), we repeat one type of measurement several times before switching to the other (e.g. XXZZ...) following [108, 115]. The number of repetitions should scale with the size of the cluster [108, 115]; here we set the number of repetitions equal to the diameter of the defect cluster.

Note that when forming super-stabilizers in the interior, we not only need to disable the data qubits connected to defective syndrome qubits, but also need to disable syndrome qubits due to defective data qubits in some cases. It is obvious that a syndrome qubit connected to no more than one active data qubit needs to be disabled. When a syndrome qubit is connected to two active data qubits but the three qubits are on the same diagonal line, it also needs to be disabled.

If a faulty qubit is too close to the boundary to be surrounded by gauge operators, it

cannot be handled by super-stabilizers. In this case the boundary of the patch needs to be deformed to exclude the faulty qubit. Although [10] addressed the boundary deformation for a surface code with a different lattice geometry, the rotated surface code has more complicated boundaries and we develop a new algorithm for handling the defects near boundary.

To illustrate a boundary deformation, four examples are shown in Fig. 3.1 (c) and (d). On the right side of (d) is a faulty syndrome qubit near a boundary of different color. Two data qubits are disabled along with it, because they are no longer included in any Z stabilizer. Then the X syndrome qubit on its right is also disabled, because it no longer has any data qubits to measure. If any of these three qubits were the faulty one, the same boundary deformation would apply. If a faulty syndrome qubit is near a boundary of the same color, as is the case on the left of (c), more qubits need to be excluded from the patch to ensure that all stabilizers on the boundary are of the same color. In particular, the neighboring syndrome qubits of different type than the boundary need to be excluded from the patch. If a data or syndrome qubit at a corner is faulty, then only one other qubit needs to be excluded (lower right of (d)). If any faulty qubit is too close to the new boundary, it must be excluded too. To the lower left of (c), such an example is shown. The faulty data qubit on the original boundary leads to an excluded region that is similar to the one on the right edge of (d). Then since a data qubit on the new boundary is faulty, the lower boundary is further deformed. Note that the second faulty qubit in this region was part of three stabilizers that remained active after the first deformation, but only the one of different color than the lower boundary is excluded in the second deformation.

The code distance d is the length of the shortest undetectable error chain on a patch of QEC code, and is equivalent to the length of the shortest X or Z logical operator. As we will show in Sec. 3.4, it not only characterizes the defect-free codes, but also serves as a primary indicator for the fidelity of defective patches. On a $l \times l$ patch, we have $d = l$ only if there is no defect; a defective patch has $d < l$. In Fig. 3.1 (a), $l = 5$ and $d = 4$ along both directions.

In (b), we have $l = 7$ and $d = 5$. In (c) and (d), $l = 9$. The code distance is 7 in (c). In (d), the distance is different along each direction: $d = 9$ vertically and $d = 8$ horizontally.

3.4 Building a device with defective qubits

In this section, we move on to the setting where the goal is to build a large FT device with an array of rotated surface code patches. We first propose a modular architecture and discuss design choices, then identify a post-selection criterion for evaluating the quality of defective chiplets.

For the simulation, we use two models of fabrication errors: one with links set to be faulty at random, and one with links and qubits both set to be faulty at the same probability. The first one models fixed-frequency transmon qubits with fixed couplers, where frequency collision is the dominant type of fabrication error. The latter models tunable transmon qubits, where links are as intricate as qubits. When using the super-stabilizers, a faulty link can be handled by disabling either of the two qubits that it connects. Faulty syndrome qubits lead to greater damage as explained in the last section, so we choose to disable the data qubit connected to a faulty link unless the syndrome qubit on the other end is already disabled.

We use a circuit-level noise model for the physical errors on the non-defective qubits, where the two-qubit gate error is p, the one-qubit gate error is $0.8p$, and the readout error is $\frac{8}{15}p$. We use standard measurement circuits for the syndromes [121].

3.4.1 A modular architecture for rotated surface code

The modular architecture we simulate in this project is an array of chiplets similar to the ones in [110]. We allocate one patch of surface code on each chiplet, as in Fig. 3.4. The qubits on adjacent chiplets can communicate via the inter-chip links (shown in dashed lines), but these links are currently $\sim 3X$ worse than on-chip links[110]. Since no patch is defined

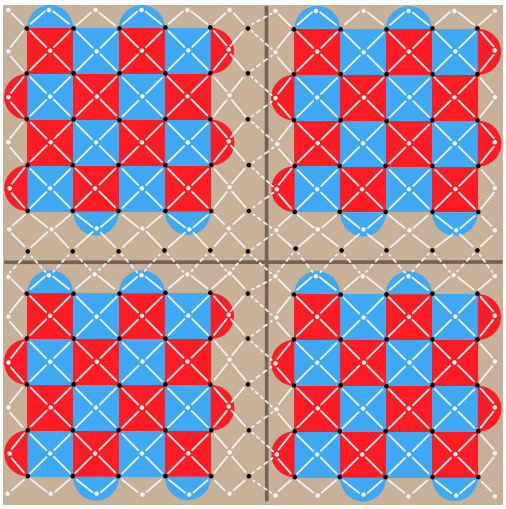


Figure 3.4: Schematic of a chiplet architecture.

across multiple chiplets, the inter-chip links are not used when a patch is idle. We assume the physical qubits on each chiplet has the grid connectivity, which is the one that naturally supports the surface code.

In Fig. 3.4, we show a design that allows one to swap the assignment of the data and syndrome qubits on a chiplet by rotating the chiplet by 180°. When a chiplet contains more faulty syndrome qubits than faulty data qubits, this will likely improve the quality of the code, because faulty syndrome qubits generally cause more significant drop in fidelity. In Sec. 3.5.1, we will evaluate how much advantage this degree of freedom translates to.

3.4.2 Post-selection criterion: assessing the quality of defective chiplets

When building a modular device, one has the opportunity to select the chips with better quality, and arrange them in a way that maximizes the fidelity. In [110], only the defect-free chiplets are kept, and then they are combined in a way that avoids frequency collisions along inter-chip links. When the goal is to support the surface code, we need different post-selection criteria. The chiplets that support higher-quality surface code patches should be kept, and they should be arranged to ensure that they can communicate at full code distance

via lattice surgery.

For the purpose of selecting and arranging chiplets, we need to find good indicators for the ability of a defective chip to support a good surface code encoding and lattice surgery. This is because in a realistic setting, it might be impractical to experimentally measure the fidelity of surface code patches encoded on each chiplet before deciding which ones to use. Experimentally testing the fidelity of lattice surgery operations between patches on different chips is even less practical, since it requires repeatedly connecting and disconnecting chiplets to iterate through different combinations. When the target logical error rate (LER) is tiny, the cost of running simulations (e.g. a memory experiment) to estimate fidelity is also formidable. This is because when the LER is small, it takes too many shots to observe enough instances of logical errors.

To investigate the relevance of different figures of merit for many sample chiplets, we need a way of evaluating the quality of individual chiplets. We adopt Eqn. (3.1) to devise one such quantity. Specifically, we look to find the exponent αd of this expression. To obtain this number, for each sample chiplet, we evaluate the logical failure rate as a function of p for values of $5 \times 10^{-4} \leq p \leq 2 \times 10^{-3}$ where logical failure rates are determined using Monte Carlo methods. This is a typical regime where the defect-free surface code is studied [41]. The value αd is the gradient of the logical failure rate shown as a function of p plotted with logarithmic axes. As such we will refer to this value as 'the slope'. We show logical failure rates plotted as a function of p in Fig. 3.6. The straight lines given in the plot indicate that we are sampling in a low p regime where Eqn. (3.1) is valid.

We explored various possible indicators including d of the defective patches, the total number of qubits that get disabled (a quantity that is generally higher than the number of faulty qubits), the size of the largest cluster of disabled qubits. The code distance of a defective surface code patch is the best indicator we find (Fig. 3.5). Although [10] suggests that defective patches are outperformed by defect-free patches with the d, their data (Fig.

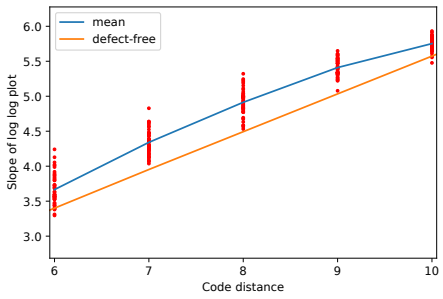


Figure 3.5: Slopes of the log-log LER v.s. p plots, from randomly sampled defective rotated surface code patches with $l = 11$. For each value of d, 50 defective patches are sampled, with the same probability for link failure and qubit failure.

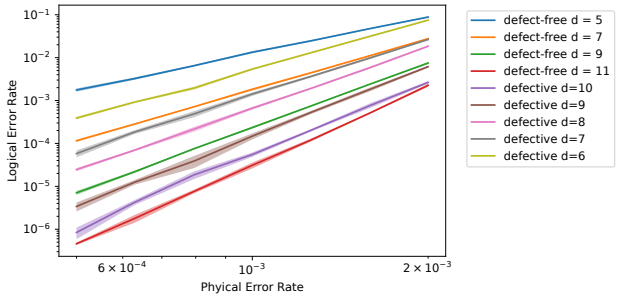


Figure 3.6: Logical error rate v.s. physical error rate at low physical error rates (5×10^{-4} to 2×10^{-3}), for defect-free patches of rotated surface code, and examples of defective patches with $l = 11$. The shaded regions represent the 95% confidence intervals for each value.

14 in [10]) only supports this claim for physical error rates $\geq 3 \times 10^{-3}$. Instead, we find that the defective patches generally have higher slopes than the defect-free patches with the same d (Fig. 3.5). This means although the defective patches perform worse than the defect-free counterparts at higher p, they generally perform better at lower p.

To explain the variation among patches with the same d, we identified a secondary indicator, the number of unique weight d logical operators. In other words, it is the number of different ways that a logical failure can occur with d physical errors. It can be evaluated efficiently with a modified version of breadth-first search on a graph where the nodes are the physical qubits on a surface code. As shown in Fig. 3.7, this helps to identify the outliers that significantly overperforms or under-performs compared to the defective patches with the same d. This indicator also helps us to understand why defective patches generally

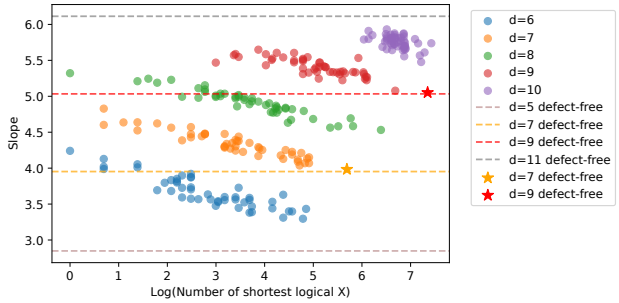


Figure 3.7: Slopes of LER v.s. p, from the same defective patches used in Fig. 3.5, grouped by d and plotted against the log of the number of shortest logical X.

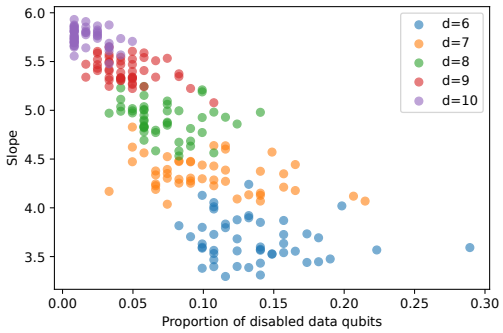


Figure 3.8: Slopes of LER v.s. p, from the same defective patches used in Fig. 3.5, grouped by d and plotted against the proportion of disabled physical qubits on a patch.

outperform defect-free counterparts with the same d: a defect-free patch has more symmetry in its shape so it has a large number of unique minimum-weight logical operators.

In Fig. 3.8 and 3.9, we evaluate two other indicators that we tested. The size of the largest defect cluster does not help predict the slope. The proportion of disabled data qubits is inversely correlated with the slope, but does not provide extra information that one cannot tell from the d.

Now, we compare our post-selection criterion against a baseline indicator, the number of faulty qubits on a chiplet. Although there is a visible negative correlation between this quantity and the slope in Fig. 3.10, it is not as effective as the indicators we choose in Fig. 3.11.

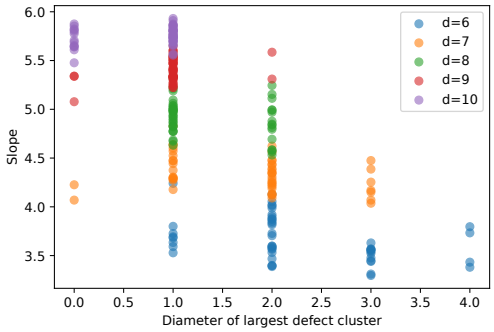


Figure 3.9: Slopes of LER v.s. p, from the same defective patches used in Fig. 3.5, grouped by d and plotted against the diameter of largest cluster of disabled qubits.

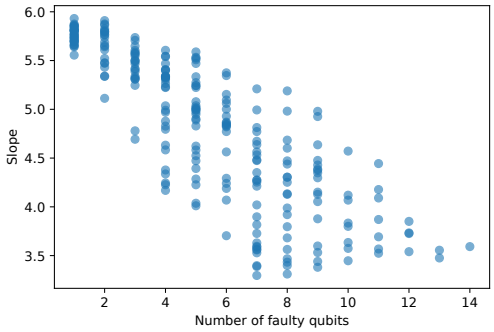


Figure 3.10: Slopes of LER v.s. p, from the same defective patches used in Fig. 3.5, plotted against the number of faulty qubits on a patch.

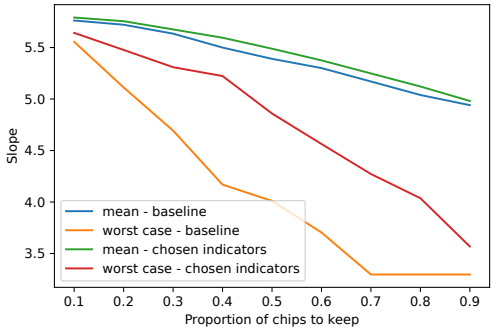


Figure 3.11: Mean and worst slopes of selected patches, when the proportion selected is varied. The baseline only uses the number of faulty qubits (Fig. 3.10); the "chosen indicators" use d as primary indicator and the number of shortest logical operators to break ties.

3.5 Impact on resource overhead

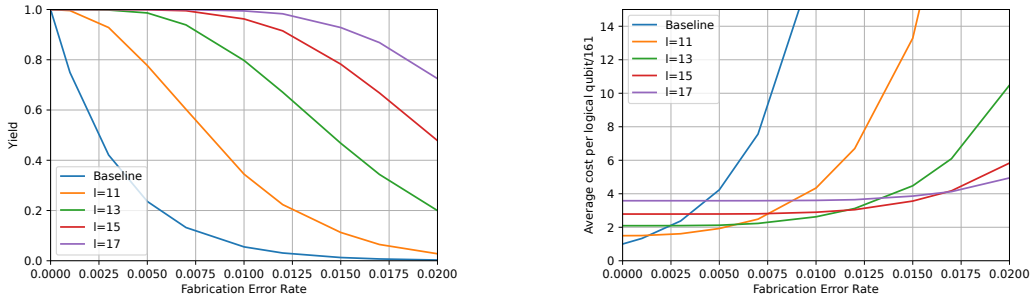
In this section, we show how design choices affect the resource overhead of building a large quantum computer to support a surface code.

3.5.1 Resource overhead and sensitivity analysis

In this section, we show how fabrication errors increase the resource overhead of the surface code, and show that a modular architecture design and super-stabilizers successfully mitigate the cost.

When the goal is to match the fidelity of the $d = 9$ defect-free patch, we have the choice of using chiplets of width 9, 11, or larger. What chiplet size is more resource-efficient? If we make larger chiplets, each patch has a higher l. Then under the same fabrication error rate, we expect more chiplets to meet the standard. With smaller chiplets, we have a lower yield, and for the baseline where $l = 9$, we cannot tolerate any defects. But each larger chiplet is made with more resources, which we quantify as the number of physical qubits. In Fig. 3.12(a), we show the yields, and in (b), we show the average number of fabricated physical qubits for a logical qubit, which is obtained by dividing the number of qubits on each patch by the yield. The simulation is run with the model that only has faulty links, and each data point is collected from a 10000-shot simulation. From Fig. 3.12(b) we can tell that below a fabrication error rate of $\sim 0.1\%$, we should choose the baseline. From $\sim 0.1\%$ to $\sim 0.6\%$ and from $\sim 0.6\%$ to $\sim 1.1\%$, we should choose l =11 and 13 respectively. When the fabrication error rate is above $\sim 1.1\%$, we should choose $l = 15$ or higher. The overhead factor of the baseline approach rises out of the figure at higher defect rates. It is 18X and 336X respectively at 1% and 2% defect rates.

When we use the model where links and qubits have the same defect rate (Fig. 3.13), instead of the link-defect-only model, the yields are lower than in Fig. 3.12 and the advantage of using larger l start from a lower defect rate. At a 1% defect rate, the overhead factor of



(a) (b)

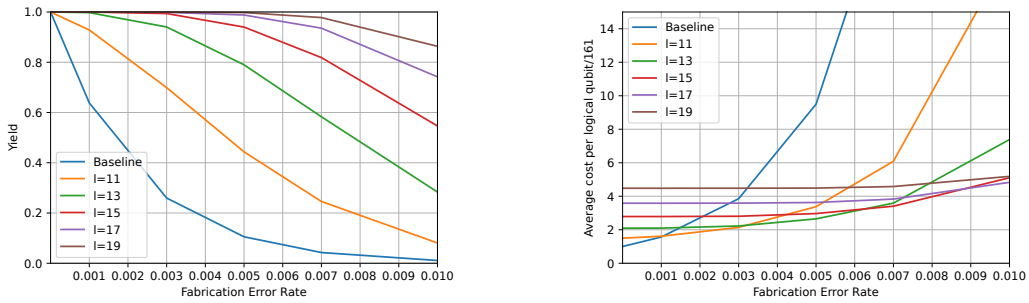
Figure 3.12: Defective links only. (a) Proportion of chiplets that support a rotated surface code patch that performs as well as a defect-free patch of distance 9, evaluated with the two metrics in 3.7. (b) As for (a) but showing the average number of fabricated physical qubits per logical qubit scaled by the number in the no-defect case.

the baseline approach ($l = 9$) is 91X.

For most of the paper, we focus on the fidelity of an individual patch. Here, we briefly discuss how certain deformations on the boundary would result in a drop in code distance during lattice surgery, and evaluate the cost of avoiding such chiplets.

Lattice surgery involves merges and splits between patches of the planar surface code. In Fig. 3.14 we show an example case where a deformed boundary only leads to a small decrease in the distance of the individual patch, but causes a larger decrease of code distance after a merge. In this example, the two merging edges are deformed at the same place. When the deformations are not aligned, there can be a greater drop in code distance after the merge. A low-distance merge has lower fidelity, so when this type of patch is used, the compiler should try to schedule lattice surgery operations on its other edges. Then, the programs would be compiled to more layers. Alternatively, one can avoid using patches with such an edge, which may result in a lower yield.

Note that we did not run simulation to compute the fidelity of lattice surgery operations between defective patches. Therefore, it is only our speculation that the code distance of the merged patch is sufficient to predict the fidelity of the logical operations.



(a) (b)

Figure 3.13: Links and qubits are assigned faulty at the same rate. (a) Proportion of chiplets that support a rotated surface code patch that performs as well as a defect-free patch of distance 9, evaluated with the two metrics in 3.7. (b) The same as for (a) but showing the average number of fabricated physical qubits per logical qubit scaled by the number in the no-defect case.

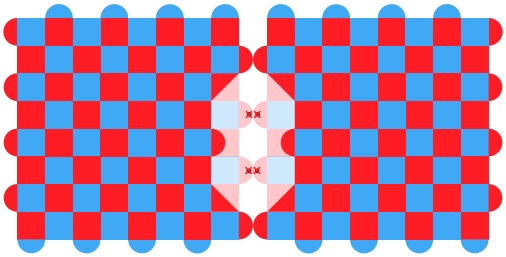


Figure 3.14: An example where the code distance drops after a merge.

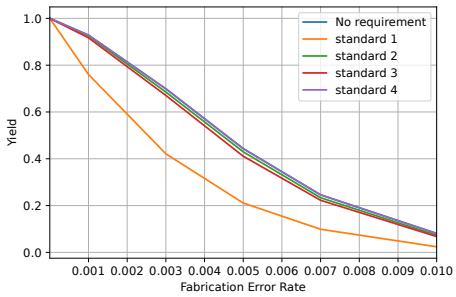


Figure 3.15: The change in yield after imposing different standards on boundaries of patches. Standard 1: No deformation on any boundary; standard 2: at least 2 boundaries of different types have no deformation; standard 3: all 4 boundaries support lattice surgery without decreasing distance; standard 4: at least 2 boundaries of different types support lattice surgery without decreasing distance.

In Fig. 3.15, we show how the yield changes after a boundary constraint is imposed. We have two types of boundary constraints for an edge of a surface code patch: (a) where we are free of deformations (in this case we do not need to form any new super-stabilizer during lattice surgery), and (b) where the total width of deformations along the edge is not enough to decrease the code distance after a merge. Then, we have the choice of imposing the constraint on (c) all four edges of a patch, or (d) on only two edges (at least one X-edge and at least one Z-edge, for the convenience of scheduling lattice surgery operations). From these, we get four different boundary constraints. The yield drops significantly only when we impose the strictest constraint (standard 1, or a and c). The drop is negligible for standard 4, when we impose (b) and (d). When we impose standard 2 or 3, the drop is visible but small. Given the results, we should apply standard 3 if we are willing to form new super-stabilizers along the merging/splitting edges; if not, we should apply standard 2.

The way we propose to allocate a surface code patch on a chiplet (in Sec 3.4.1) allows the freedom to swap the assignment of data/syndrome qubits by a 180°rotation. We observe a small improvement in yield when we have such a freedom (Fig. 3.16). The improvement is smaller in the regime with a higher number of faulty qubits - when l is large or the fabrication error rate is high. This is because when there are defects on both links and qubits, a patch

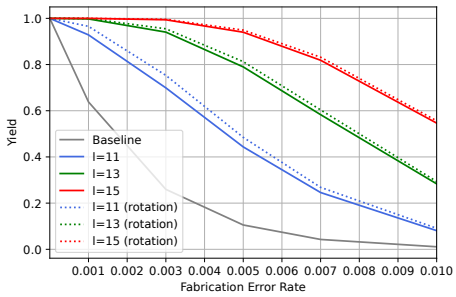
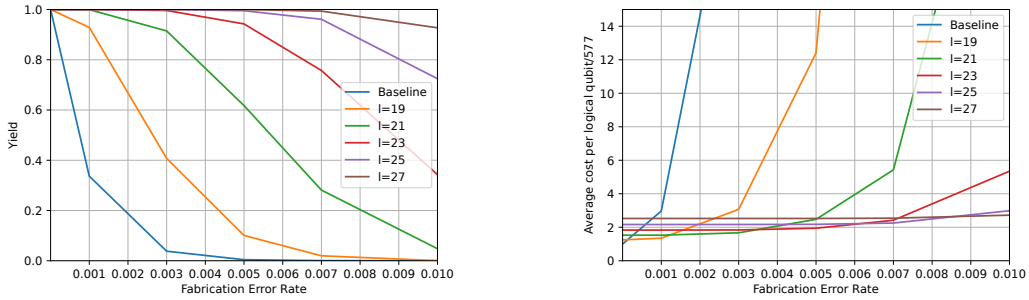


Figure 3.16: Improvement in yield when there is freedom to rotate the chiplets. Links and qubits are assigned faulty at the same rate.

tends to have more faulty data qubits than syndrome qubits since we handle each defective link by disabling the end with data qubit. Also, this holds with a higher certainty when there are more defects. Some techniques to reduce leakage errors involve swapping data and ancilla qubits [83], which might not work well with this design. Handling leakage errors is outside the scope of this paper, however.

In Fig. 3.17, we show the cost of making higher-quality logical qubits. For this set of simulations, instead of matching the fidelity of the $d = 9$ defect-free rotated surface code, the goal is to match the $d = 17$ defect-free code. The trends we observe are qualitatively the same. Note that at 1% defect rate, the factor of resource overhead from the $l = 17$ defect-intolerant baseline is over 56000X. In fact, when the defect rate is fixed, the resource overhead increases exponentially with the number of physical qubits in a logical qubit.

On Fig. 3.12b,3.13b, and 3.17b, if we take the minimum of all curves at each fabrication error rate, we obtain the minimum extra resource overhead (due to defects) that can be achieved by the chiplet architecture considered in this work. In Fig. 3.18, we show how this factor is affected by the fabrication error rate and the target code fidelity. When the fabrication error model consists of defective links only, the curves for different target fidelity coincide. It is $\sim 2X$ at a 0.5% defect rate, and below $3X$ at 1% defect rate. When we model both defective qubits and links, the curves coincide at low defect rate and diverge a small amount at higher defect rate. The factor of overhead is $\sim 3X$ at a 0.5% defect rate and 5X



(a) (b)

Figure 3.17: Yields for larger chiplets. Defective links only. (a) Proportion of chiplets that support a rotated surface code patch that performs as well as a defect-free patch of distance 17, evaluated with the two metrics in 3.7. (b) The same as for (a) but showing the average number of fabricated physical qubits per logical qubit scaled by the number in the no-defect case.

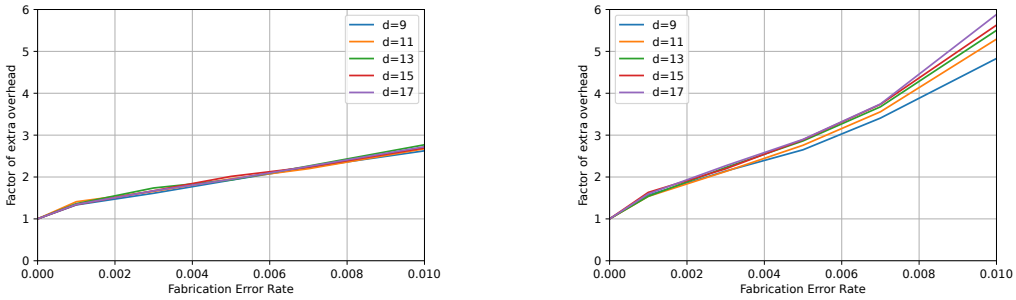
to 6X at 1% for the range of fidelity targets we set.

Fig. 3.18 shows that, even with a method to implement QEC in the presence of fabrication errors, it is still important to reduce the defect rate by adopting improved designs for qubits or fabrication. Limiting the factor of overhead to below 2X requires a defect rate below $\sim 0.5\%$ for the link-defect only model, and below $\sim 0.2\%$ for the model with both defective links and qubits.

3.5.2 Limit of the monolithic architecture

The results in Sec. 3.5.1 demonstrate that the ability to implement the surface code on defective grids is necessary for containing its resource overhead, by comparing against a baseline design that uses modular chiplets but only accepts defect-free surface code patches. What if we choose a monolithic architecture but accept defective patches?

On a monolithic device holding an array of surface code patches, not all the good patches can be used - they must also be connected at the very least. In Fig. 3.19, we show the results from simulating a monolithic device holding a square array of logical qubits. Each patch of logical qubit is randomly assigned to be good/bad, and then we search for the



(a) Defective links only.

(b) Defective links and qubits.

Figure 3.18: The extra resource overhead due to defects, for different target logical error rates. The y-axis is the average number of fabricated qubits for a logical qubit, scaled by the number in the ideal no-defect case. The target logical error rate is the fidelity of the defect-free rotated surface code of code distance d.

largest connected component (LCC) of the good patches. The x-axis of the figure is the total number of patches on the device, while the y-axis shows the expected size of the LCC of good patches and the expected number of good patches. The results show that not all the good patches on a monolithic device are connected unless the yield is as high as 90%.

We conclude that the monolithic architecture imposes a higher requirement on the yield. A very high proportion of the patches must be good, otherwise the good patches are disconnected by the bad ones. For example, in the setting of Fig. 3.13, the yields from the most resource efficient approaches are 63.7%, 69.9%, 79.0%, 81.9%, and 74.2% respectively at defect rates 0.1%, 0.3%, 0.5%, 0.7% and 1%. In all 5 cases, some of the good patches will be disconnected. When the yield is 63.7%, on a device with 10 by 10 patches the expected size of the LCC is 48.3, which means 24% of the good patches (63.7) are not in the LCC. Note that evaluating the size of LCC of good patches on a monolithic device does not fully reflect the disadvantage of the monolithic architecture. Most importantly, the good patches in the same connected component on a monolithic device are connected more sparsely than those on a modular device solely made with good patches. Furthermore, the modular design provides the flexibility to reduce the poor match of edges that may result in low-quality

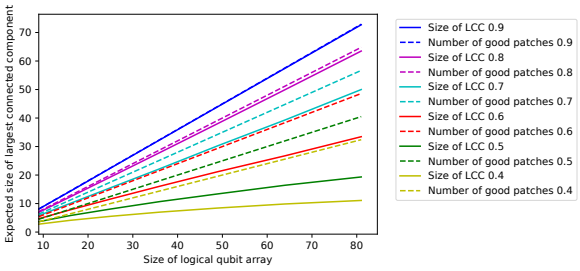


Figure 3.19: Limit of a monolithic architecture. The figure shows the expected size of largest connected component of the good patches on a square array of surface code patches allocated on a monolithic device, where each patch is randomly assigned to be good/bad. The label of each line is the proportion of the good patches.

lattice surgery. This optimization is not available on a monolithic architecture.

The analysis above focused on the connectivity problem that arises when we only use the patches that meet the requirement on a monolithic device. If all patches are used, the ones that do not meet the performance requirement will lower the application fidelity, as shown in Section 3.5.3.

3.5.3 Resource overhead and fidelity estimation for an example application

In Section 3.5.1, the quantity we use for evaluating each approach is the factor of extra resource overhead relative to the ideal defect-free case. In this section, we estimate the resource cost and application fidelity of an example application in the presence of defects.

The application we choose is Shor's algorithm applied to 2048-bit integers, whose implementation with surface code (in the no-defect setup) is optimized and analyzed in [47]. It requires a $226 \cdot 63$ grid of distance-27 surface code patches, and about 25 billion surface code cycles, according to [47].

In Table 3.1 and 3.2, we show cost estimates for building a modular device that supports the application, at a defect rate of 0.1% and 0.3% respectively (on both qubits and links). To compute the cost of the super-stabilizer approach, we use the steps in Section 3.5.1, and find the optimal choice of l that minimizes the factor of resource overhead for a target code

distance of 27. The no-defect baseline is for the ideal case where no defects arise, and the defect-intolerant baseline is for the design choice to only use defect-free chiplets. The results show that the defect-intolerant approach makes the already tremendous resource requirement for the algorithm orders of magnitude higher, while the super-stabilizers help lower the factor of resource overhead to a small number (45X better for the 0.1% defect rate and more than 10^5X better when the defect rate is 0.3%). This example also demonstrates the importance of reducing the defect rate. When the defect rate is increased from 0.1% to 0.3%, the cost increases by 40% even when the super-stabilizers are applied.

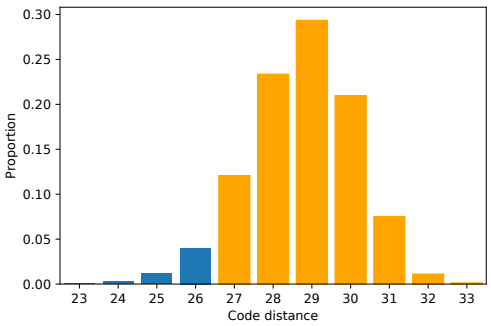
	No-defect	Defect-intolerant	Super-stabilizer
l	27	27	33
Yield	100%	1.4%	94.5%
Overhead	1	71.32	1.58
Qubits	2.1×10^7	1.5×10^9	3.3×10^7

Table 3.1: Resource estimation for building a device that supports a $226 \cdot 63$ grid of distance-27 surface code patches, for a defect rate of 0.001 on both qubits and links. *No-defect* is the ideal setting without fabrication defects, not an approach to handle defects. Both the *defect-intolerant* and the *super-stabilizer* approaches here post-select chiplets to make a modular device. *Overhead* is the factor of resource overhead, determined by the yield and the size of each chiplet; *Qubits* is the total number of physical qubits fabricated for the application.

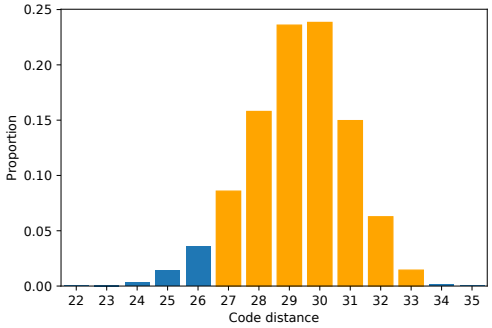
	No-defect	Defect-intolerant	Super-stabilizer
l	27	27	39
Yield	100%	2.7×10^{-6}	94.6%
Overhead	1	3.67×10^5	2.21
Qubits	2.1×10^7	7.6×10^{12}	4.6×10^7

Table 3.2: Same as Table 3.1 but for a defect rate of 0.003.

The fidelity of a large-scale fault-tolerant application can be roughly estimated with the topological error rate, as in Section 2.13 of [47]. We follow their method to estimate the fidelity of the application, assuming the physical gate error on the device is 10^{-3}. In the calculation, we account for the code distance distributions for the adapted surface code



(a) $l = 33$, defect rate at 0.1% for both links and qubits.



(b) $l = 39$, defect rate at 0.3% for both links and qubits.

Figure 3.20: Distribution of code distance. In orange: proportions of patches with $d \geq 27$; in blue: proportions of patches with $d < 27$.

patches (see Fig. 3.20). The distributions are each obtained from a sample size of 10000. Recall that based on the results in Section 3.4.2, code distance is the most important indicator for the performance of an adapted surface code patch. Furthermore, the logical error rate of the adapted surface code is generally lower than that of the defect-free patch with the same code distance. Therefore, using code distance to estimate the performance of each patch does not underestimate the failure rate for the super-stabilizer approach.

Estimates for the application fidelity are shown in Table 3.3 and 3.4. Note that the device from the ideal no-defect case (all patches are exactly distance 27) would have a fidelity of $\sim 73\%$. In the case where we use a modular device with super-stabilizers, all the patches are at least distance 27 and most of them have larger code distances. Therefore, the estimated application fidelity is higher, albeit at a higher resource cost than the ideal no-defect case.

We compare our approach against two baselines while holding the resource overhead constant. The first baseline's goal is to build a modular, defect-free device from defective qubits. For this baseline, we need to lower the resource overhead to be the same as the super-stabilizer approach. We do this by reducing the size of the chiplets. The factor of resource overhead for building this modular defect-free device from defective qubits (relative to the distance-27, ideal no-defect case) is 1.12 for $d = 15$ and 2.09 for $d = 17$, at a defect

	baseline1	baseline2	Modular & super-stabilizer
l	$15 \sim 17$	$33 \sim 35$	33
Overhead	1.58	1.58	1.58
Estimated fidelity	0	79.9%	88.5%

Table 3.3: Application fidelity estimated with the topological error. Baseline1: modular, defect-intolerant. Baseline2: monolithic, uses super-stabilizers to handle defects.

	baseline1	baseline2	Modular & super-stabilizer
l	$11 \sim 13$	$39 \sim 41$	39
Overhead	2.21	2.21	2.21
Estimated fidelity	0	76.1%	91.7%

Table 3.4: Same as Table 3.3 but for a defect rate of 0.003.

rate of 0.1%. To match the resource overhead of our approach (1.58), one would use a mix of $d = 15$ and $d = 17$ defect-free patches. However, these code distances are insufficient for the application; they both result in an estimated fidelity of effectively 0. Furthermore, when the defect rate is 0.3%, the defect-intolerant baseline can only afford patches of distance 11 and 13, which is even farther below the requirement.

The second baseline is the monolithic device with the super-stabilizers applied. There is no post-selection of chiplets for the monolithic device, so its resource overhead is lower if the same l is used. In order to match the resource overhead factor of our approach, we increase l for a proportion of the patches. For the case with a 0.1% defect rate, we keep 53% of the patches at $l = 33$ and use $l = 35$ for the rest. For the defect rate= 0.3% case, we keep 47% of the patches at $l = 39$ and expand the rest to $l = 41$. Our calculation shows that even with the increased code distance, the expected failure rate of this baseline is $\sim$ 1.8X and $\sim$ 2.9X higher than the modular, super-stabilizer approach for the two cases. This is because the logical qubit patches with $d < 27$ contribute to the error rate on the monolithic device, negatively impacting fidelity, but they are discarded in the modular case.

3.6 What counts as a fabrication error

So far we have been using a simple model for defects, where each qubit or link is labeled directly as "faulty" or "non-faulty". In real life, there are scenarios where it is not clear whether a qubit should be viewed as faulty. For example, on a device with a average 2-qubit gate fidelity of 99.9%, should we disable a qubit that only supports 2-qubit gates with 97% fidelity? If the cutoff is set too high, we lose too many physical qubits and suffer a decrease in code performance; if the cutoff is too low, the inferior qubits will also damage the code.

To identify cutoff fidelity values for labelling a qubit faulty, we need to compare the logical performance when we keep the faulty qubits against the results from disabling them.

We use the stability experiment [46] instead of the more standard memory experiment that we used in the previous sections. While the memory experiment quantifies how well a logical observable is maintained by the code, the stability experiment evaluates how well a logical observable can be moved (a capacity that is needed for logical operations). As explained in [46], measurement errors can't cause logical errors in memory experiments except by creating confusion that hides the key data errors. Since errors caused by a faulty qubit look like repeated measurement errors, the memory experiment is unable to show the damage of faulty qubits.

In Fig. 3.21, we show the results from a stability experiment where the data qubit in the center of a $d = 5$ surface code has higher error rate than the rest. We set the value of p, the two-qubit error rate of the worse qubit, to values from 5% to 15% (the other errors on it scale accordingly). Then, we compare the results against the one from disabling the worse qubit and using super-stabilizers around it. The figure shows that when p of the bad qubit is above $\sim 10\%$, we should disable it regardless of the quality of the other qubits. When p is below 5%, it is preferable to keep it unless the error rate on the other qubits is below the range in the plot. Finally, when $p = 8\%$, we should disable the qubit if the error rate on the other qubits is below $\sim 0.45\%$.

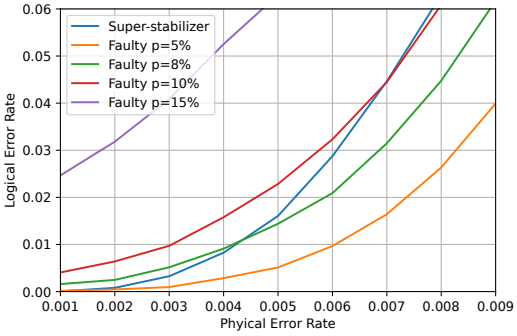


Figure 3.21: Stability experiment results from keeping/disabling a bad data qubit on a $d = 5$ surface code. The x-axis is the physical error rate of the good qubits.

3.7 Related work

To the best of our knowledge, [110] is the only prior paper that also advocates for a modular quantum chiplet design to mitigate fabrication defects. They analyze how to reduce the resource overhead in the pursuit of making a defect-free device. In contrast to [110], which does not specialize in any particular application, our research specifically targets quantum error correction. This focus allows us to utilize some defective chiplets, significantly reducing the resource overhead compared to [110].

Our method for adapting the surface code to defective qubit arrays builds upon earlier work on super-stabilizers [10, 115, 108]. However, prior studies did not establish a post-selection criterion applicable to modular chiplets, nor did they perform an analysis of the associated resource overhead. [10] proposed to correct errors that occur near to defects using super-stabilizers. The idea was built from [114, 113] where super-stabilizers are constructed to correct for an idealized noise model that introduces loss errors. As the authors of [10] point out, it is not clear if their measurement schedule to read out super-stabilizers will give rise to a threshold. [10] also introduced a protocol for adapting the surface code to arbitrary defect distributions, but the boundary deformation in the protocol only applies to the *unrotated* surface code, which uses $\sim 2X$ more physical qubits per logical qubit for a commensurate code distance with the rotated code used in this paper. Our simulator

includes a new algorithm for deforming boundaries due to the more complicated boundary of rotated surface code. Another difference is that we implement the shells proposed in [115] to mitigate clustered defects.

Some other methods have also been proposed to handle faulty qubits. Nagayama et al. [89] also formed large stabilizers around the defects. They use SWAP gates to collect all the syndrome information onto one qubit, while we adopt the approach that takes the product of gauge operators. Wu et al. [129] developed an algorithm to adapt surface code to devices with sparse connectivity such as the current IBM devices. According to our correspondence with them, their method is more suitable for highly symmetric lattices and is less suitable for handling arbitrary defect distributions.

We focus on static defects in this paper, but transient events such as cosmic rays could result in temporary defects. There are some strategies [118] that are specifically designed for transient defects on QEC code. [108] recently considered producing shells around large clusters of transiently defective qubits introduced by cosmic rays. This work identified the importance of varying the schedule of super-stabilizer measurements for clustered defects.

3.8 Conclusion

Building a large device with modular chiplets provides the flexibility to throw away unwanted chiplets and arrange the rest. Such flexibility is crucial for scaling up quantum devices to support QEC in the presence of fabrication defects. In this work, we implement an automated method to adapt a rotated surface code to a defective grid and generate syndrome measurement circuits. Then, we run numerical simulations to identify effective indicators for assessing the performance of defective chiplets relative to defect-free chiplets. With these indicators, we evaluate the resource overhead of implementing an array of logical qubits with different target fidelity and under different defect rates. We also analyze how the overhead is affected by factors like the chiplet size. We found that with modularity and the super-

stabilizers, the increase of resource overhead due to defects can be limited to a small factor, which is orders of magnitude better than the defect-intolerant baseline.

We have focused on the design that allocates one patch of a surface code on each chiplet. Dividing each patch onto multiple chiplets would increase the flexibility in post-selection. However, since inter-chip links are currently $\sim 3X$ worse than on-chip links[110], this decision might increase the logical error rates. Whether further division of chiplets can reduce the overhead could be an interesting subject for future work.

CHAPTER 4

SPATIALLY PARALLEL DECODING FOR FAULT-TOLERANT QUANTUM LOGICAL OPERATIONS

4.1 Introduction

Given the error rates experienced by quantum computers, quantum error correction (QEC) is necessary for running large-scale quantum applications. QEC protects quantum information by encoding each logical qubit in multiple physical *data* qubits, and using another set of physical *syndrome* qubits to detect errors on the data qubits. In each measurement cycle, syndrome qubits extract parity information from the data qubits on which they act. This parity information is then sent to a classical decoder that decodes the syndromes and reports a correction.

Here we focus on the surface code [34, 41, 65], a popular QEC that tolerates relatively high physical noise, supports relatively easy logical operations, and only requires nearest-neighbor grid connectivity. In recent years, academic and industry labs have experimentally demonstrated small instances of a surface code logical memory [5, 69, 139]. While an offline decoder — applied after the experiment has completed — is sufficient for such demonstrations, applications with nontrivial information processing will require real time decoding [41].

A requirement for real time decoding is that the throughput of the decoder match the rate of syndrome measurements, which avoids an exponential backlog of data [120]. And the requirement can be quite strict: each syndrome measurement cycle on a superconducting device can be completed in $\sim 1\ \mu s$ (921 ns in [5]). The strict timescales have persuaded the community to explore hardware accelerators for the task, such as Field Programmable Gate Arrays (FPGAs) [29, 123, 79], Application Specific Integrated Circuits (ASICs) [30], or on-chip SFQ-based superconducting digital circuits [100, 58, 122]. While their software counterparts — designed to run on general-purpose CPUs [56] — offer more flexibility, hard-

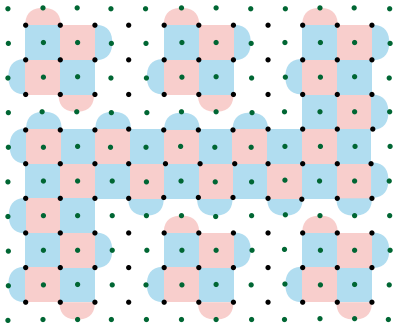


Figure 4.1: A merge operation in lattice surgery, specifically, a logical $Z \otimes Z$ measurement.

ware decoders promise higher throughput, deterministic execution, and tighter integration with the rest of the control system [11].

Whichever decoder one chooses, a single monolithic instance will inevitably struggle to meet the decoding demands that arise during multi-qubit logic gates. For surface codes, lattice surgery [60, 40] is the most efficient way to perform multi-qubit logical operations. When lattice surgery is performed on two or more (possibly distant) patches of logical qubits, they are merged into a single patch for multiple measurement cycles before being split apart; a snapshot of the procedure is shown in Figure 4.1. It's worth noting that current state of the art decoders can only meet the throughput demands on a patch that is the size of an individual logical qubit up to distance 20 to 30 [11]. Worse, their throughput is inversely proportional to the size of the patch [56]. Meanwhile, lattice surgery requires merging multiple such patches [77] and decoding the merged patch.

Dividing the decoding task into overlapping windows is a promising approach [109, 119, 15] to manage this scalability challenge. For instance, prior work [109, 119] leverages temporal parallelism by processing the syndrome data from many measurement rounds with *temporally* parallel windows. This limits the growth of syndrome backlog when the inner decoder is slower than syndrome generation in terms of throughput. But this strategy doesn't mitigate the large spatial window that each parallel decoder must cover. In [109], the authors introduce the idea of dividing the decoding problem in both time and space, but do

not analyze its feasibility. In [15], the authors leverage a different kind of spatial parallelism, where independent subgraphs arising from the logical circuit are decoded in parallel. This addresses the spatial challenge, but requires a dynamic choice in where parity information is routed. We suspect this dynamic routing may prove complicated for hardware decoders to realize in practice.

Here, we explore *spatially parallel windows*. To the best of our knowledge, ours is the first analysis of a decoding scheme that is both capable of handling large patches that arise during logical operations *and* compatible with practical system level constraints of hardware accelerators. As alluded to in [109], we employ the strategy of dividing the decoding task into multiple overlapping windows, and assign a decoder module to each window. The inner decoder that operates on an individual window can be any real time decoder; our scheme is agnostic to the choice. The important point is that these decoders can be coordinated to output a correction when a merged patch spans multiple windows. This technique works because we only need to protect logical information up to the distance of the isolated logical qubits. This allows us to overlap windows with local views and focus on resolving the disagreements at their seams. The decoders are scheduled to run during different time steps, so that they only need to resolve with their neighbors before and after they run.

A motivating principle of our work is that various design choices need to be carefully balanced to ensure that a scheme with spatially parallel windows can meet the requirements of real time decoding. A hardware constraint specific to the problem is that each window needs to be pre-assigned to an area on the device. This is because real time decoders that operate on individual windows, regardless of the specific implementation, require hardware accelerators with fixed positions. Furthermore, our work emphasizes the importance of avoiding larger windows than necessary, since large windows challenge the scalability of inner decoders.

We begin by demonstrating how to configure spatially parallel windows in a manner that

(1) is compatible with hardware accelerators like FPGAs and ASICs, and (2) can handle the larger merged patches that arise during lattice surgery operations. The decoding scheme is scalable, in the sense that its speed and resource requirements are independent of the number of patches that are merged or how far they are apart on the device. Then we examine the factors influencing the performance, focusing specifically on two key aspects: accuracy and throughput. We proceed to analyze how to achieve a balance between these requirements. Finally, we estimate the size of the individual code patches this scheme can support, assuming one uses an FPGA-based inner decoder.

By performing numerical simulations and analyzing the mechanisms through which spatially parallel windows lead to extra decoding errors, we identify that the size of the windows and the width of the overlapping areas (*buffer width*) are important factors that determine the accuracy. To maintain the fidelity of the logical qubits, the size of the windows cannot be smaller than the individual patches that encode the logical qubits. The buffer width, however, is a key variable in the trade-off between accuracy and throughput. Enlarging the buffer results in larger windows, which subsequently leads to lower throughput (or significantly raises the requirement on computing resources). Conversely, when the buffer is too narrow, it compromises the accuracy of the decoding. We find that the optimal choice of buffer width not only depends on the size of individual code patches, but also depends on the level of physical noise on the device. This can be explained by examining the terms in the logical error rate expression, especially the entropic factors. At a modest physical noise level, the buffer width should be between one-half and two-thirds of the width of an individual patch of code.

The throughput of the decoding scheme is determined by its slowest component. To identify the bottleneck, we separately estimate the speed of both the inter-window communication and the decoding modules. Our findings indicate that the inter-window communication will not become the bottleneck, even when the width of the windows exceed 100. This means

the overall throughput of the decoding scheme meets the requirement for real time decoding if and only if each inner decoder module runs sufficiently fast on its window. We discuss the scalability of two state-of-the-art real time decoders [79, 11] when applied to individual windows.

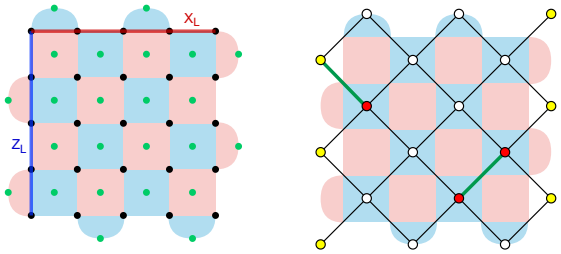
Although we focus on surface code in this work, our results are also relevant to other local codes, e.g. color codes.

4.2 Background

4.2.1 Surface code, lattice surgery, and decoding

A square patch of surface code encodes one logical qubit in a $d \times d$ grid of data qubits, where d is the code distance, and uses $d^2 - 1$ syndrome qubits for the stabilizer measurements. A distance-5 example is shown in Figure 4.2a. The stabilizer measurements project errors on the data qubits into Pauli X, Y or Z errors. Each X (Z) stabilizer measures the X (Z) parity of the data qubits that it acts on, and detects the Z (X) errors on them since the X and Z operators anti-commute. An X (Z) syndrome qubit reads 1 if an odd number of its data qubits are affected by a Z (X) error. In this case we say the stabilizer is flipped. A flipped syndrome readout is also called an *anyon* or a *defect* in literature. A Y error can be viewed as the product of an X error and a Z error. It can be detected by both types of the stabilizers.

An X (Z) logical operator, $\hat{X_L}$ ($\hat{Z_L}$), is a product chain of X (Z) operators on a set of data qubits that connect the two X (Z) boundaries. Examples of them are shown in Figure 4.2a. The logical operators do not flip any stabilizer. Note that if a chain of operators is equivalent to a linear combination of the stabilizers, it does not flip any stabilizer either, but it acts trivially on the logical qubit. We define the code distance for X (Z) errors, d_X (d_Z), as the weight of the shortest X (Z) logical operator. It is the minimum weight of a nontrivial X



(a) (b)

Figure 4.2: (a) A square patch of surface code. The red and blue faces are the X and Z stabilizers, respectively. The blue and red lines mark $\hat{Z_L}$ and $\hat{X_L}$, the Z and X logical operators. (b) An example decoding graph for the X errors. The vertices are the Z stabilizers, augmented by the virtual boundary node (in yellow). The stabilizers that are flipped are marked in red, and a minimum weight matching is denoted by the green edges.

(Z) error chain that cannot be detected by the code. The code distance d is the minimum of d_X and d_Z. If a device has biased noise, e.g. more Z errors than X errors, then a rectangular patch with different d_X and d_Z could be used to further suppress the dominant type of errors.

Lattice surgery implements logical multi-qubit Pauli measurements via the merging and splitting of patches, which can be used for performing logical gates. For instance, when two patches are merged along boundaries that coincide with their Z logical operators, the value of $Z \otimes Z$ can be inferred from the product of the Z stabilizers spanning the boundary. X Pauli products are measured similarly, while measurements that involve the Y operator can be done with twist-based lattice surgery [78] or an alternative protocol [24]. When logical qubits are separated in space, the merge operation can be facilitated by a long ancilla patch as in Figure 5 in [78].

While all errors chains with weight smaller than d are detectable by the code, only the ones with distance no more than $\lfloor \frac{d}{2} \rfloor$ are guaranteed to be correctable. The decoding of syndrome is performed on a decoding graph (Figure 4.2b). For the surface code, independent X and Z decoding graphs can be used for correcting X and Z logical errors. On a decoding graph, each edge represents an error mechanism that is detectable by its vertices. An edge

between two adjacent Z stabilizers represents an X flip on the data qubit shared by them. An edge connected to the virtual boundary node represents an error that is only detected by one stabilizer.

For simplicity, Figure 4.2b only shows a 2D decoding graph for one round of syndrome measurement. The full decoding graph that a decoder works with has a time dimension because measurement errors also need to be represented. Such a graph can be viewed as multiple copies of Figure 4.2b stacked on top of each other. If two vertices represent the readouts of the same stabilizer from adjacent time steps, they are connected by an edge that represents a flip on the syndrome qubit.

A popular technique for decoding syndromes from a surface code is minimum-weight perfect matching (MWPM) [34], which identifies the lowest weight error patterns on a given decoding graph. Sparse Blossom [56] and Fusion Blossom [131] are recent, fast software implementations of this technique. Union-Find (UF) [32] is another popular technique, and can be viewed as an approximation of the Blossom algorithm [36, 37] that implements MWPM [130]. It is known to have better runtime complexity and slightly lower accuracy than MWPM.

4.2.2 Real time decoding with FPGA

For concreteness, we explore the implications of using an FPGA for the inner decoder. FPGAs offer an attractive alternative to general-purpose CPUs for tasks with strict timing requirements and modest programmability requirements. An FPGA is made up of a "fabric" consisting of a great many components whose behaviors are individually programmable, whose input and output can be flexibly routed among other components, and whose timing is uniformly specified so as to support easy latching of inter-component signals. These properties imbue FPGAs with excellent parallelism and pipelinability, enabling them to achieve excellent throughput on high-bandwidth tasks. The primary trade-off for these

abilities is that the FPGA fabric is almost always programmed "once and for all"—that is, the individually programmable components nonetheless have their behaviors fixed for the lifetime of an application.

Because surface code syndromes arrive in a stream at approximately $(d^2 - 1)$ Mbps, decoding is a strong candidate application for an FPGA [79, 29, 123]. However, the constraints of FPGA programming bear directly on the decoding problem:

- Scalability: An individual FPGA enjoys a fixed set of components, making it suitable for decoding only up to a certain size. Meanwhile, arbitrarily long swaths of surface code may appear during lattice surgery protocols.

- Communication: The high-speed communication afforded by the lock-step evolution of FPGA components is somewhat lost when connecting two separate fabrics together (e.g., over ethernet). Accordingly, programmers need to keep tight control over inter-fabric communication to keep it from dominating the runtime of an application. Additionally, local connections between components are set out at "compile time", sometimes coming at significant layout cost.

- Fixed gateware: Since FPGAs only permit modest on-the-fly reprogramming, care must be taken to match these limited abilities with the changing decoder requirements of a surface code patch undergoing lattice surgery.

Deploying an FPGA to accomplish decoding requires accommodating each of these constraints.

4.3 Related work

In recent years, there has been a surge in research on real time decoding for QEC. Helios [79] is an FPGA-based UF decoder. When acting on a surface code with distance d, it achieves a sublinear average time complexity per cycle, at the cost of $O(d^3)$ hardware resources.

The work demonstrated an implementation operable up to $d = 21$. Riverlane [11] recently implemented a UF decoder on both FPGAs and ASICs and reported results for up to $d = 23$. Astrea [123] and LILLIPUT [29] output the same solution as a MWPM decoder for surface code up to $d = 7$ and $d = 5$, respectively. These can be implemented on FGPAs. AFS proposes to implement the UF decoder on ASICs [30]. There are also real time decoders with SFQ-based superconducting digital circuits [100, 58, 122]. Some decoders use hierarchical decoding [100, 25, 111].

A few recent works divide the decoding task into overlapping windows [109, 119, 15], an approach that is also employed in our work. In [119] and [109], the authors primarily address parallelization in time, which helps contain the backlog when the throughput of the decoder does not match the rate that the syndrome is generated. In [109], the authors also includes a discussion on dividing the decoding problem in space but do not explore the consequences. The information passed between neighboring windows is similar, whether they are temporally or spatially parallel. However, temporally and spatially parallel windows apply to different problems and so have different constraints and goals. For example, [109] does not investigate how a narrow buffer would compromise the accuracy of decoding.

A paper from PsiQuantum [15] is the only prior work that targets lattice-surgery style fault-tolerant blocks. Its focus is different from our work though. It prioritizes minimizing the latency of a software implementation and does not address “further hardware and systems considerations that are relevant". However, we put more emphasis on the constraints of hardware accelerators, and consider throughput instead of latency as the more important measure of speed for real time decoders. For example, they develop algorithms that take lattice surgery blocks as input, then assign windows, while we explicitly require that windows have pre-assigned positions so that they are compatible with hardware accelerators. Both papers include numerical results on how the buffer width affects decoding accuracy, but we provide more analysis and also study how the influence depends on the physical noise level.

4.4 Framework of the decoding scheme

In Section 4.2.2, we discussed three constraints of an FPGA implementation. Our protocol keeps these constraints foremost in mind: our starting premise is that a single decoder handles a fixed (small) window; we explicitly analyze the single burst of communication needed from one stage of decoding to the next, which takes place over a fixed subset of a nearest-neighbor topology network; and we limit our reprogrammability requirements to a simple form of "masking", where read and write operations are individually dis/allowed at given cells.

In this section we explain key aspects of the design. We first show how the decoder modules should be connected and why it is necessary for them to overlap in buffer regions. Then we discuss the metrics of the decoding scheme and the design choices that influence them. Finally we introduce a configuration that is compatible with the hardware constraints.

4.4.1 Connecting decoder modules

The time that it takes for a monolithic decoder to process a long patch of code scales at least linearly with the area of the patch. (Exceptions like look-up-table-based decoders have exponentially high storage overhead so they only work for very small patches [29]). Spatially parallel decoding circumvents this by dividing the decoding task into multiple processes. In order to avoid frequent communication between the processes, we do not consider fine-grained schemes where each (ancilla or data) qubit is assigned its own decoder module. We only consider coarse-grained parallel schemes where each decoder module works on an area that is similar to an individual patch of logical qubit like Figure 4.2(a). We refer to the area that a decoder module acts on as a *window*.

It might seem resource efficient to divide the device into non-overlapping windows, and run decoders on each window simultaneously. However, this will either completely sacrifice the accuracy of decoding, or require high communication overhead between neighboring

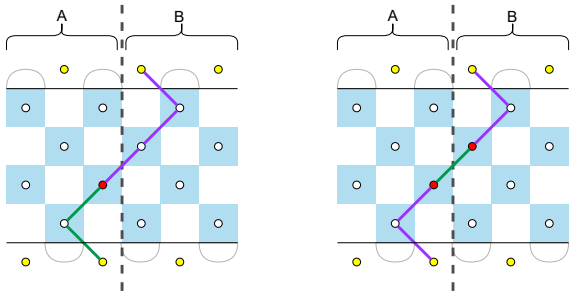


Figure 4.3: A segment from a longer patch of surface code, divided into 2 non-overlapping windows by an artificial boundary (dashed line). The optimal corrections are shown in green and the suboptimal ones in purple. (a) One flipped syndrome in window A. The suboptimal correction will be selected if matching to the artificial boundary is allowed. (b) An error flips two syndromes, one in each window. The suboptimal correction will be selected if matching to the artificial boundary is disallowed.

decoder modules. If no communication is allowed between non-overlapping windows, the decoder has two choices when acting on one window: (1) it allows matching across the boundary with neighboring windows, (2) it does not allow such matching. The examples in Figure 4.3 show that either case leads to easy decoding failures. In case (1), the flipped syndrome in Figure 3(a) would be matched to the boundary between the two windows, while it should be matched to the lower boundary. In case (2), the flipped syndromes in Figure 3(b) would be matched to the top and bottom boundaries, while they should be matched to each other. Both examples result in logical errors.

In order to maintain the accuracy of decoding and avoid high communication overhead, one can take the approach of using overlapping windows and applying decoder modules on neighboring windows in different *layers* (time steps). We will use terms *rough boundary* and *smooth boundary* in the explanation. On a decoding graph, a boundary with edges to the virtual boundary node is termed rough, while a boundary with no edges out of it termed smooth. These boudaries can refer to either ones that already exist in the global decoding graph (e.g. the top and bottom boundaries of Fig. 4 are rough), or an artificial one that arises from partitioning the patch into multiple windows (e.g. the vertical boundaries in

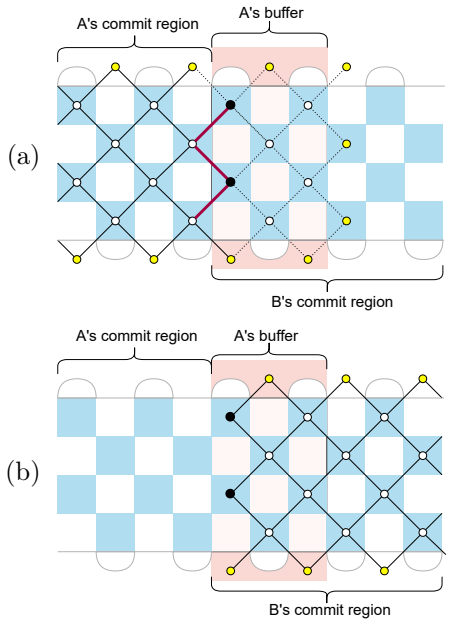


Figure 4.4: Decoding graphs for X errors, used by window A and B respectively. The buffer is marked in pink. In (a), the red edges are the ones that connect the nodes in A's commit region to the buffer. The two nodes where artificial defects could be introduced are marked by solid black dots. Corrections along the dotted edges are on nodes in the buffer, and are not committed by window A.

Fig. 4 are rough and smooth respectively). These terms should not be confused with the rough and smooth boundaries of a surface code, which are synonymous with boundaries that absorb strings of Z and X flips, respectively.

Figure 4.4 illustrates this approach for a segment of the X decoding graph taken from a longer patch. As shown in shown in Figure 4a, window A has a rough boundary on its right side. The edges that connect to this boundary are derived from those that straddle window A in the global decoding graph. In the first layer, only window A is acted on by a decoder. After it obtains a correction, the decoder only commits the part that acts on the nodes in A's *commit region*, marked by the solid edges in Figure 4a. The edges with dotted lines are in the buffer between the windows A and B, and corrections in this region are not applied by window A.

The only communication between windows A and B happens when A finishes decoding.

And the only information A passes to B is the *artificial defects*, the nodes along the left (smooth) boundary of B that are flipped by corrections in the commit region of A. The artificial defects are marked with solid black dots in Figure 4. After B learns about these defects, it performs corresponding flips on the nodes along its left boundary and begins decoding, as shown in Figure 4b. This ensures consistent corrections across the windows.

Unlike in window A, the decoding graph of window B (Figure 4b) does not have an artificial rough boundary with neighboring windows. This is true for any window scheduled in the final layer. In the X decoding graphs shown in Figure 4, there are still natural rough boundaries at the top and bottom; however, these would be smooth in the corresponding Z decoding graph and so this graph would have no rough boundaries. When a decoding graph has no rough boundaries, the decoder will fail to output a correction if the syndrome it receives contains an odd number of flipped nodes. In other words, there is a *lone defect* that cannot be matched. When this happens, we terminate decoding and report that a logical error has occurred. This failure is due to suboptimal decoding in previous layers.

4.4.2 Metrics

Throughput is one of the most important metrics for a real time decoder. It should match the rate at which syndromes are generated so that the syndrome backlog will not grow exponentially, exhausting storage space and slowing down logical operations [120]. When the decoder has enough throughput, it is sufficient to apply the sliding window technique [62] along the time dimension. When the inner decoder is not fast enough, a temporally parallel decoding scheme can mitigate the backlog problem. But temporal parallelization alone is not an ideal solution for the long patches that arise from lattice surgery, which will require more layers of temporally parallel windows. This will slow down the logical clock rate further and also incur more hardware costs if the decoder is implemented with hardware accelerators.

Accuracy is a critical metric for any decoding scheme. When multiple logical qubits

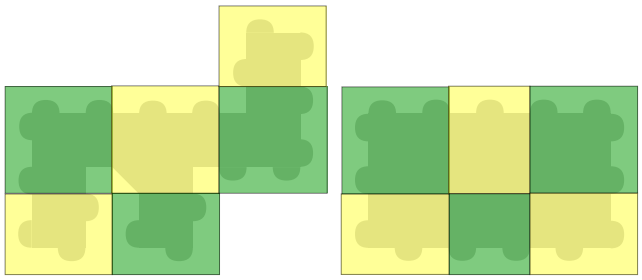
undergo a logical operation, they should ideally maintain the same level of fidelity as when they are isolated. As we will show in Section 4.5, this requires sufficiently large windows and buffers.

Latency is another metric of the decoder's efficiency. It measures the total delay between the time that a syndrome measurement cycle finishes and the decoder outputs a correction. It is particularly important when performing conditional operations, like gate-by-measurement. The latency of the decoder determines how long a logical qubit needs to idle before the conditional operation can be executed. This is because the value that conditions the operation is in part determined by the decoder's corrections. While it doesn't have a strict limit like throughput, a large latency is still undesirable because it leads to an increase in the space-time volume of the computation [25, 31]. Specifically, the time of the computation increases linearly with the latency, and to protect the logical information, the code distance should be increased logarithmically to compensate. However, the scheme we consider only has a latency of a few cycles. The logical error rate per cycle is roughly proportional to $(100p)^{(d+1)/2}$ [39]. At $p \sim 0.1\%$, increasing d by 2 would suppress the logical error rate by $\sim$ 10X, which is enough for overcoming the latency.

The latency of the spatially parallel windows is determined by (1) the latency in each component of the decoding scheme, which is already reflected in the throughput, and (2) the number of layers that the windows are divided into. To reduce the latency, it is preferable to use a window configuration with a small number of layers. This is one of the considerations in the following subsection.

4.4.3 Window configuration for hardware decoding

For a software implementation of the spatially parallel windows, one can dynamically and flexibly adjust the configuration of the windows during the computation, given the lattice surgery operations at each time step. However, for real time decoding using a hardware



(a) (b)

Figure 4.5: (a) A patch with a tree structure is 2-colorable. (b) A patch that arises during a $Y \otimes Y$ logical measurement. Not 2-colorable.

accelerator like an FPGA or ASIC, the positions of the windows and the links between them should ideally be fixed. Here, we seek a window configuration that is compatible with hardware accelerators and also accommodates general lattice surgery operations.

Besides having fixed window positions and links, the window configuration and mapping/compiling strategy should respect three other constraints to ensure the accuracy of decoding. First, the commit regions of the windows in the same layer must be disjoint. Unlike windows in different layers (e.g. the two windows in Figure 4.4), they cannot share data qubits. This is because no information is shared between windows in the same layer, while the information necessary for making consistent corrections is passed between neighboring windows in different layers. Second, the windows should not have smaller size than an isolated patch of code (e.g. Figure 4.2a). Thirdly, when a patch spans multiple commit regions, the intersection of the patch with each commit region should not be too small. The second and third points will be further explained in Section 4.5.

To reduce the latency of the decoder, the windows should be arranged in a small number of layers. Can a two-layer configuration meet the requirements? For the large patch in Figure 4.1, it suffices to use a 2-colorable checkerboard configuration of windows. This is because the patch has a tree structure. See Figure 4.5a for a window configuration that accommodates a multi-patch merge of this type (only the commit region of each window is shown). In this example, although the commit regions with the same color overlap at

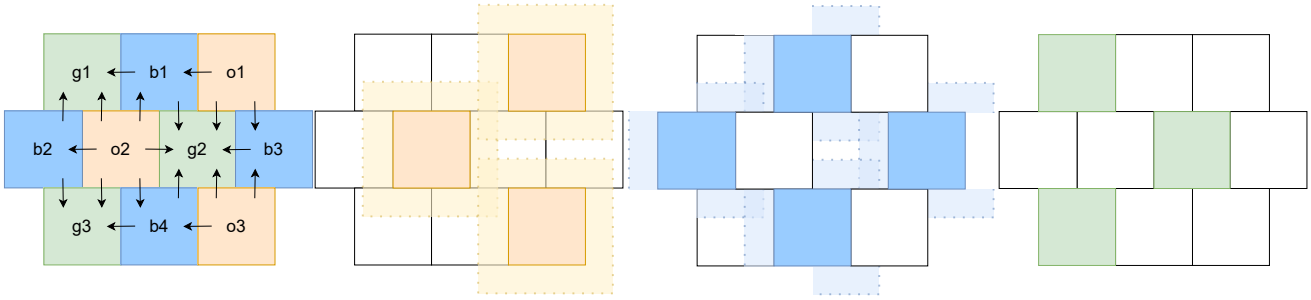


Figure 4.6: A window configuration with staggered squares. (a) Only showing the commit region of each window. Each arrow represents a link that communicates information on artificial defects. (b-d) The windows in the first, second, and third layers, respectively. The buffers are shown in lighter color and surrounded by dotted edges. The windows on the last layer do not have buffers. The buffer width in this figure is chosen to improve readability.

corners, their committed corrections do not conflict because no active qubits are at these corners.

However, not all patches from lattice surgery operations have a tree structure similar to that in Figure 4.1 and Figure 4.5a. These examples only involve X and Z logical measurements. But when a lattice surgery operation involves a Y measurement on a logical qubit, the patch may need to be touched on more than one X or Z edge, which may result in a patch that cannot be accommodated by a checkerboard pattern. For instance, Figure 4.5b shows a merged patch that arises during a $Y \otimes Y$ logical measurement [23].

To accommodate general lattice surgery operations we propose to use a 3-coloring of staggered squares (Figure 4.6). It is a simple configuration that satisfies all the constraints mentioned above. The size of each commit region is the same as the size of an individual patch of code. If the rows of windows are not staggered then 4 colors are required, which leads to an unnecessary increase of latency.

4.5 Logical error rates

In this section we study the logical error rates (LER) when using spatially parallel windows. We begin by describing the simulation setup on a rectangular surface code patch, and then study logical errors along the short and long edges of the patch. We then move on to simulations and analysis for a more general setup, five windows stacked in two rows. We also discuss how the results influence design decisions (e.g. buffer width) and introduce constraints on mapping.

4.5.1 Methodology for simulations

We perform quantum memory simulations, where each shot includes d cycles of stabilizer measurements. We use Stim [45] for simulating stabilizer circuits along with a circuit-level noise model that includes single-qubit, two-qubit, and measurement errors characterized by a single parameter p. We use PyMatching2 [56], a state-of-the-art implementation of the MWPM decoder, both as the inner decoder applied to individual windows and as the baseline global decoder. Although evaluation of the throughput would require a real time inner decoder, PyMatching2 is sufficient for simulations that focus on logical error rates and syndrome densities.

For Sections 4.5.2 and 4.5.3 we use a rectangular surface code patch that would arise when merging two square-shaped patches of distance d during lattice surgery. Our goal is to maintain the fidelity of the individual patches, and we study the impact of the buffer width and physical noise p on this fidelity. As such, we use the simplest setup (Figure 4.7) with just two overlapping windows, with window A applied before B. This configuration is such that the square-patches are the same size as the commit regions of the windows.

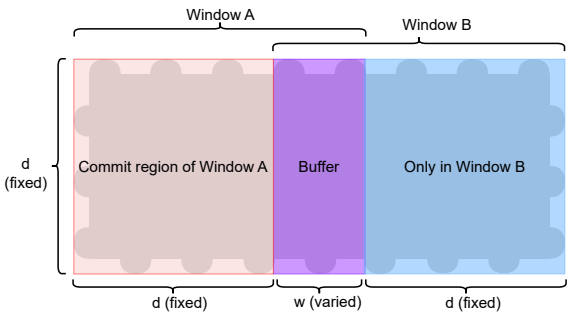


Figure 4.7: Setup for the numerical simulations.

4.5.2 Logical errors along the short edge

In a memory experiment, a logical error occurs if an X or Z logical operator is flipped after applying both the noise and the correction. We first study the impact of logical errors along the short edge of Figure 4.7, with code distance $d = 15$. We only use one of the X and Z (global) decoding graphs: the one with rough boundaries on the top and bottom edges. For this setup the logical operator should be a horizontal string that connects the left and right edges of the entire patch.

In Figure 4.8 we plot the LER against w, the buffer width (e.g. Figure 4 has $w = 3$). The figure shows that when the buffer width is small, the accuracy of the parallel decoder is orders of magnitude worse than its global counterpart. It also shows that increasing w closes the gap. In fact, the gap diminishes before the buffer is grown to match the length of the short edge, which implies that w need not be as large as d to maintain accuracy. This is beneficial for throughput, since large buffers invariably decrease the speed of each decoder module or place tighter requirements on the underlying hardware.

So how large does w need to be? From Figure 4.8 we observe that the optimal choice of w depends on the physical noise p. At $p = 0.1\%$, the 95% confidence intervals of the LERs from parallel and global decoding overlap when $w \geq 10$. At $w = 10$ and $p = 0.1\%$, the ratio of parallel to global LER is ~ 1.32. But at $p = 0.5\%$, the ratio is below 1.3 when $w \geq 6$, and it is only ~ 1.05 at $w = 10$. This shows that one might need a larger w at lower p to

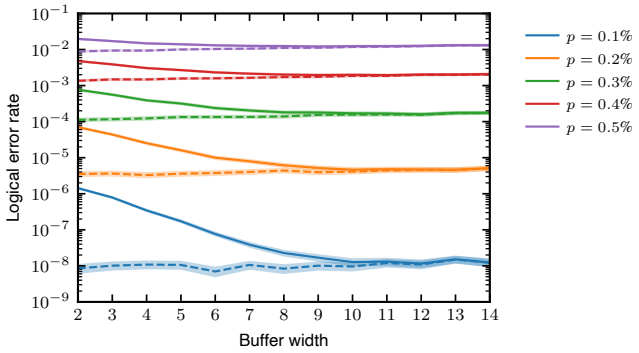


Figure 4.8: Logical error rate v.s. buffer width, using the setup in Figure 4.7 with $d = 15$, counting the logical errors along the short edge (that connect the top and bottom boundaries). The dashed lines are the baseline results from using a *global* decoder on the same underlying patch. The shaded regions indicate the 95% confidence intervals. Each data point at $p = 0.1\%$ is obtained with 8B shots.

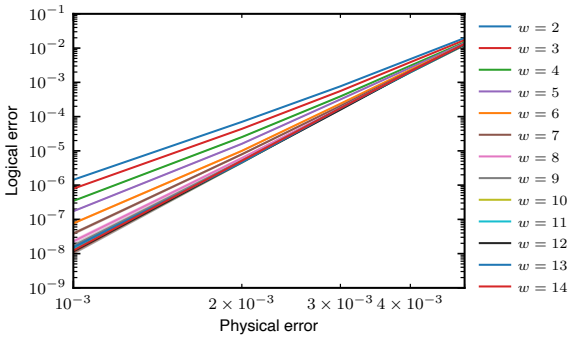
match the accuracy of global decoding.

In Figure 4.9, we provide an another view of the results from Figure 4.8. In Figure 4.9a we plot the LERs against p on log scale. The slope of a line in the plot shows how quickly the LER is suppressed when p decreases, and is a proxy for effective code distance. When we plot the slope against w in Figure 4.9b, we see that the benefit of growing w diminishes as w increases.

To facilitate understanding of the numerical results, we also study error strings that confuse the parallel decoder but can be corrected by a global decoder. In Figure 4.10 we show examples of such error strings for $d = 15$, $w = 9$ and 3. The example error strings have lengths (l) 6 and 4, respectively. Since l is no more than $\lfloor \frac{d}{2} \rfloor$ (which is 7) for both strings, they can be corrected by a global decoder.

However, in Figure 4.10a, window A can match the left end of the string to the top boundary (and commit this), and the right end to the rough boundary outside of the buffer (tentatively). Such a matching consists of 6 edges, the same as the correct one, so window A will output one or the other with equal probability. When window B is in action, it only sees the right end of the error string, and matches it to the bottom boundary. Since one end of

(a)



(b)

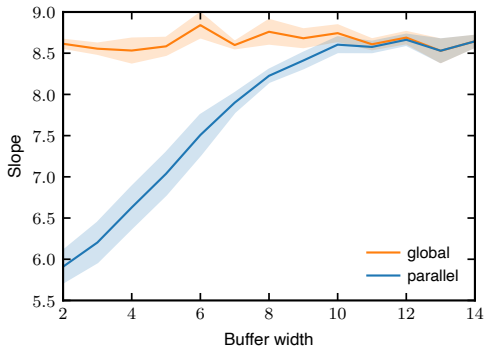
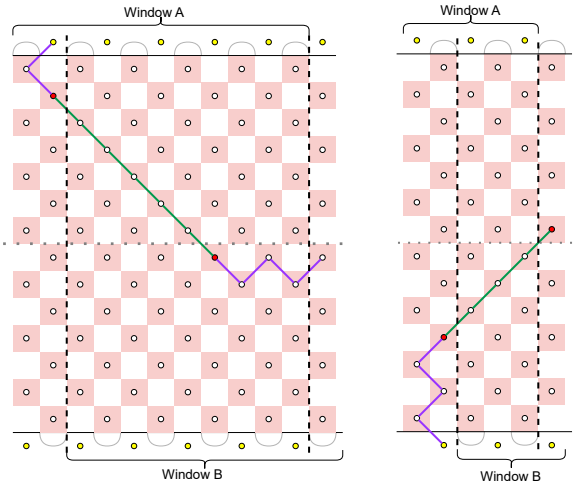


Figure 4.9: The data in Figure 4.8 represented differently. (a) Logical error rate v.s. physical error rate, for parallel decoding with different buffer width w. (b) The slopes of the lines in (a) and (b), plotted against w.



(a) (b)

Figure 4.10: Example error strings that confuse the parallel decoder but can be corrected by a global decoder. Left: $w = 9$. Right: $w = 3$. In each example, the region between the two dashed vertical lines is the overlap of windows A and B. Window A is decoded first. The flipped stabilizers are in red. The error strings are in green. The purple edges are wrong decoder outputs by A. The dotted horizontal line marks the middle of the patch.

the error string is matched to the top boundary and the other is matched to the bottom, the correction flips the horizontal logical operator while the noise does not. Thus the operator is flipped at the end and we have a logical error. In Figure 4.10b, window A can match the left end of the error string to the bottom boundary. The right end of the string is invisible to A, so A does not need to act on it. Such a matching is again the same weight as the actual error string. Then window B only sees the right end of the error string and matches it to the top boundary.

How long do these strings need to be? For a connected error string (that is correctable by some global decoder) to result in a logical error along the short edge, it should satisfy three conditions:

1. One end of it is matched to the top boundary and the other is matched to the bottom, in the committed matching

2. Exactly one end of it is in the commit region of window A. If neither end is in the commit region of A, no correction will be committed by window A, and the error is entirely decoded by B as in the case of a global decoder. If both ends are in the commit region of A, then the corrections to both defects are committed by A.

3. In window A, the end that is not matched to the top/bottom boundary is matched to the rough boundary out of the buffer. If both ends are matched to the same boundary, we do not get a logical error. If they are matched to opposite boundaries by A, the global decoder would also do such a matching.

Let l be the weight of the error string (its number of edges). The right end of the error string can be just above or just below the middle of the patch, and the left end can be just within the commit region of A. The weight of the wrong matching in A, described above, is the sum of the distance from the right end to the rough boundary out of the buffer, and from the left end to the top/bottom boundary. Assuming an odd d, the first part is

$\max(w+1-l,0)$, and the second part is $\max(\frac{d-1}{2}+1-l,0)$, for a diagonal error string similar to the ones in Figure 4.10. The wrong matching might be output by decoder A if it has no higher weight than any correct one. Then for $d=15$, a string with $l=4$ $(5,6,7)$ can confuse a parallel decoder with $w\leq 3$ $(w\leq 6, w\leq 9, w\leq 12)$. Note that when $w\geq d-2$, a connected error string that can confuse the parallel decoder is also long enough to confuse any global decoder.

The lengths and the quantity of error strings help explain the LER. The chance of having a weight l error is roughly $(1-p)^{n-l}p^l$, where n is the total number of edges in the 3D decoding graph (we say *roughly* because the probability associated with an edge in the decoding graph is not exactly p — the circuit-level noise model adds complications). The LER can be expressed as $P=\sum_{l=l_{\min}}^{n} N_{\text{fail}}(l)(1-p)^{n-l}p^l$, where $N_{\text{fail}}(l)$ is the number of weight-l errors that cause logical error and $l_{\min}$ is the minimum weight of an uncorrectable error (which does not need to be a connected string).

When p is low, the LER is usually dominated by the term from $l_{\min}$. This is why $l_{\min}$ is an important quantity, whether in the decoder-specific context in this paper or the decoder-independent context where it is equivalent to $\lceil\frac{d}{2}\rceil$. The range of p we use for the numerical simulations includes 0.1% which is already low. However, our simulation results cannot be fully explained by the min weight of uncorrectable errors. For example, Figure 4.10b has a $l_{\min}$ of 4 with parallel decoding, but it has more than two orders of magnitude lower LER than a distance 8 surface code decoded globally (which also has $l_{\min}=4$) under the same p of 0.1%.

The value $l_{\min}$ alone does not fully explain the simulation results because the entropy of error chains sometimes play a more important role in determining the LER at modest physical noise [13, 19]. When the buffer is just narrow enough for an error of weight $l_{\min}$ to cause a logical error, as in both examples in Figure 4.10, the entropic factor $N_{\text{fail}}(l_{\min})$ is a tiny value. The entropic factors of higher order terms in P are significantly larger. When p is

not low enough, the disparity in error probability is not enough to offset the difference in the entropic factors, then the main contribution to the LER is from the higher order terms. If the LER from the parallel decoder is dominated by terms with weight no less than $\lfloor \frac{d}{2} \rfloor$, then it will be close to the LER from the global decoder, even when the two decoding schemes have different $l_{\min}$. This explains the results in Figure 4.8.

4.5.3 Logical errors along the long edge

For a rectangular patch of surface code with a long edge and a short edge, the logical errors along the long edge constitute only a small portion of the total LER when a global decoder is used. Here, we check that spatially parallel windows preserve this property.

We again use the setup in Figure 4.7, and show the results of numerical simulations in Figure 4.11. This time, we observe that the LER gap between parallel and global decoding stays constant as the buffer is grown. We note a distinction between Figure 4.11 and Figure 4.8. As the buffer width w increases, the code distance increases along the long edge but remains constant along the short edge. This is why the LER from the global baseline decreases as w grows in Figure 4.11 but remains almost constant in Figure 4.8. We also find that the LER from parallel decoding coincides with the one from applying the global decoder to a smaller patch that is the same size as window A.

Along the long edge, we must consider error strings like the one in Figure 4.12. Window A in this figure has code distance 9, the entire patch has distance 13, and the green error string has weight 5. The string is not long enough to confuse the global decoder on the entire patch, but is long enough to confuse a window. In particular, window A would prefer the incorrect purple matching to the green one, because it only has weight 4. Then window B, which only sees the right end of the error string, has no choice but to match it to right rough boundary of its decoding graph. The result is a logical error along the horizontal edge.

Our findings about the LER along the long edge are conclusive: as long as the horizontal

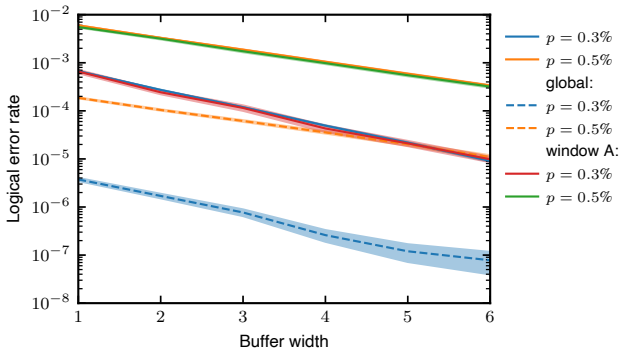


Figure 4.11: Logical error rate v.s. buffer width, using the setup in Figure 4.7 with $d = 7$, counting the logical errors along the long edge. The dashed lines are the baseline results from using a *global* decoder on the same patch. The results from applying the global decoder on a patch that is the size of window A are also shown.

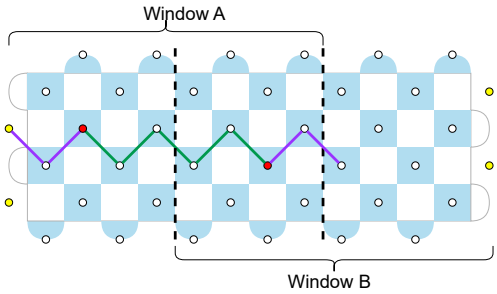


Figure 4.12: An example error string that causes a logical error along the long edge, when the parallel decoder is used. As in Figure 4.10, window A is applied before B, the buffer is between the vertical dashed lines, the error string is in green, and a wrong decoder output by A is in purple.

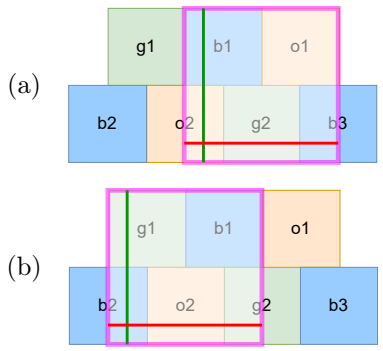


Figure 4.13: Setups for simulations with 5 windows stacked in 2 rows. The area of the underlying patch is enclosed by the pink square. The vertical and horizontal logical operators are in green and red. In (a) the two original patches that are merged are labelled b1 and o1, while in (b) they are labelled g1 and b1

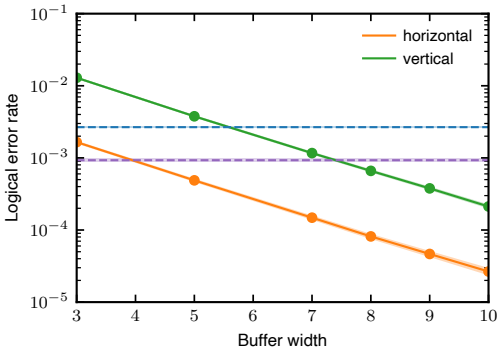
dimension of window A in Figure 4.7 is longer than d (which will be the case due to the buffer), the increased LER from logical errors along the longer edge is negligible when compared to the increase from the errors along shorter edge. Therefore we do not need to consider this type of logical errors when choosing w.

4.5.4 Generalizing to 2D configurations

As mentioned in Section 4.4.3, some lattice surgery operations involve thicker merged patches. This section investigates whether the window configuration in Figure 6 works on these patches, and the buffer width requirement for decoding on these patches.

We imagine a setting where two vertically-stacked square patches, each with the same size as a commit region in Figure 6, are first deformed into rectangles with longer horizontal edges, so that they can perform $Y \otimes Y$ logical measurement [78, 23]. Then the top and bottom rectangles are merged. We perform simulations with the 2 settings shown in Figure 13, where the underlying patch is at the top right and top left corners of the windows in Figure 6, respectively. For baseline we take the LER of the two isolated patches, either before all the logical operations (when they are squares), or right before the merge (when they are longer rectangles).

(a)



(b)

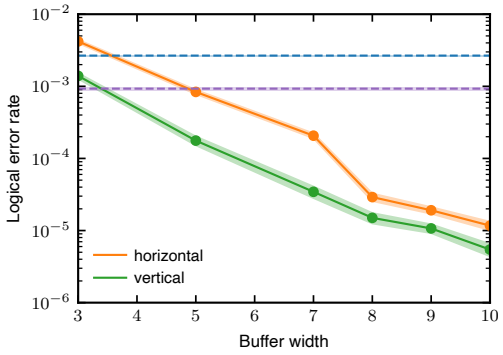


Figure 4.14: LER v.s. buffer width at $p = 0.4\%$ over 15 rounds, using the setups in Fig. 13a and 13b respectively, with horizontal and vertical logical operators. The size of each smaller square (e.g. the commit region of b1) is chosen to be 15×15. The blue (purple) baseline is the LER of two isolated patches, each with size 15×29 (15×15), taken over 15 rounds and with logical errors along the edge with $d = 15$.

The simulation results in Figure 14 shows that, in this setting, the LER of parallel windows is suppressed exponentially with w, and is below the baseline before w is grown to more than half the commit region width. Compared to the linear case presented earlier, the parallel windows can be seen to make use of the extra distance present during this logical operation — the buffers extend in both directions rather than one direction. The difference between Figure 14a and b is due to the order that the windows act. In (a), the case with vertical logical operator has higher errors because the intersection of the merged patch and the commit regions of o2 and b3 have short horizontal edges, and o2 and b3 are scheduled before their neighbor g2. This does not apply to (b) because in the lower row of (b), the middle window acts first.

4.5.5 Implications

We conclude that a sufficient buffer width w is needed for achieving high accuracy. Since linear settings (Figure 7) have larger buffer width requirements than multi-row settings (Figure 13), it suffices to choose w based on Section V-B. Section 4.5.3 shows that the size of the windows (or specifically, the intersection of a patch with a window) also bound the fidelity that can be achieved. This is acceptable as long as a window is wider than an isolated patch of code. One mapping/compiling constraint should be noted though. When a patch spans multiple commit regions, and its intersection with one of the windows is too small, the type of logical errors in Figure 4.12 can be large. This is the case in Figure 13a, where the intersection of the patch with the commit region of o2 has width $d/2$. When w is at least half of d, o2's total width is at least d, so the contribution from this type of error is not too large. However, the intersection of a patch with a commit region should not be smaller than half the commit region.

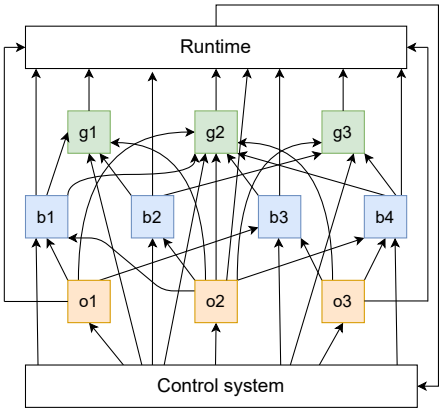


Figure 4.15: A flowchart of the parallel decoding scheme, the colored squares correspond to the windows in Figure 4.6.

4.6 Throughput

In this section, we shift the focus to throughput and study how large the window size can grow before the throughput limit is reached. We quantify throughput by the average time that it takes for a decoder to process one round of measurements. Since each surface code measurement cycle takes $\sim 1\ \mu s$ on a superconducting qubit-based quantum computer, we set it as the standard that a real time decoder should meet.

Figure 4.15 shows a flowchart of a real time decoder using the spatially parallel windows. Each window receives information from the control system measuring the qubits and its neighboring windows from previous layers. After decoding, each window sends the committed corrections to the runtime and its artificial defects to neighboring windows in later layers.

This results in a pipelined scheme where the throughput is limited by the slower of two components: the inter-window communication or the inner decoder. In this section, we first show that the inter-window communication will not be a bottleneck for an FPGA-based implementation, then discuss the scalability of inner decoders and how it affects the maximum window size that an FPGA implementation can have. To achieve larger window sizes than supported by FPGA-based inner decoders, one may turn to more resource efficient ASICs [11].

4.6.1 Inter-window communication

The throughput of the communication links depends on the number of bits that it takes to transmit information on the artificial defects. As explained in Section 4.4.1, the only information that a decoder passes to its neighbor in a subsequent layer is the artificial defects. The length of this message is determined by (1) the number of bits required to represent an artificial defect and (2) the expected number of artificial defects that arise during decoding.

Quantity (1) depends on the number of different locations for artificial defects. Between two neighboring windows A and B, this is proportional to the area of $A_{\text{commit}} \cap B$, the intersection of window B and the commit region of A. Suppose we use the staggered square configuration, where the commit regions are $d \times d$ and d cycles are processed at once. Then, between any pair of neighboring windows, the number of possible artificial defect positions is $\sim d^2/2$ for either the X or the Z decoding graphs. This translates to $\lceil \log_2(d^2/2) \rceil$ bits for representing each artificial defect — 9 bits for d ranging from 23 to 32, and 12 bits for d from 65 to 89.

To study quantity (2), we look at the distributions of artificial defects recorded from the simulations in Section 4.5.2 (Figure 4.16). We found that the distributions are independent of w and heavily dependent on p. Distributions from different values of p can be fit with the Poisson distribution with different means (λ), with the mean of the distribution roughly proportional to p. When we compared the data from Section 4.5.2 and Section 4.5.3, we find that the mean of the distribution is also proportional to the area of $A_{\text{commit}} \cap B$. This is not surprising: p is proportional to the chance that an artificial defect arises at each possible location, and the area of $A_{\text{commit}} \cap B$ is associated with the number of such locations.

Given p and the area of the face where the artificial defects lie, we can estimate the mean count of artificial defects by extrapolating from our simulation results. For example, at $p = 0.1\%$ and $d \sim 30$, the mean count of artificial defects is ~ 5. Then for each of the X and Z decoding graphs, it takes about $5 \times 9 = 45$ bits to transmit the artificial defect

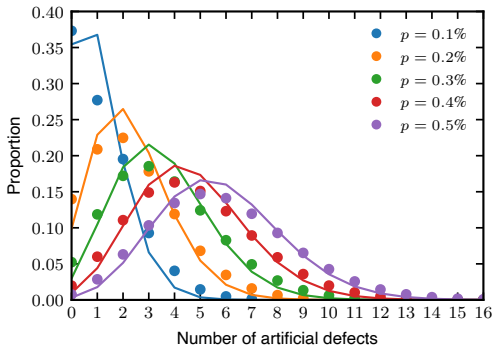


Figure 4.16: The number of artificial defects, from the same simulations as Figure 4.8. The curves are fit to the Poisson distribution.

information, without further optimization. This sums up to ~ 90 bits in total, for a link between two neighboring windows.

For an FPGA-based implementation, each copy of the inner decoder is implemented on one FPGA and acts on one window. Therefore, the inter-window communication is between FPGAs. For reference, the communication between two Xilinx FPGAs using a standard high-speed serial communication link is 54 clock cycles, plus 1 clock cycle per 64-bits [101]. On FPGAs, clock cycles of 4 ns (250 MHz) can be routinely achieved, even at high resource utilization. For windows with $d \times d$ commit regions arranged as in Figure 4.6, we have shown that, at $p = 0.1\%$, the information passed between two neighbors is at most ~ 94 bits for $d \leq 30$, if d rounds of syndrome is processed at once. That means each of these inter-FPGA links will take roughly 56 clock cycles, or ~ 224 ns if running at 250 MHz. Divided by d, the cost of communication is only 7 ns per measurement cycle, well below the requirement of 1 μs. We also calculated the communication overhead for larger d, and found that it decreases with d in the range that is relevant. At $d = 151$, the average cost of an inter-window communication is below 3 ns per measurement cycle. We performed the same calculation at a high physical noise of $p = 0.5\%$, and found that the overhead is 44 ns per round at $d = 5$ and steadily drops to below 9 ns at $d = 151$. This is because when we divide the link latency by d, the contribution from the constant latency (54 clock cycles) decreases

as d increases, which outweighs the increase in the other term that is proportional to d^2. We conclude that the inter-window communication does not limit the size of windows.

4.6.2 Inner decoders

The throughput of the inner decoder depends on the size of the windows and the choice of decoder. Since spatially parallel windows can be layered over any inner decoder, and the development of real time decoders is still an active field, there is no clear answer for the maximum size of the window. Nonetheless, we can discuss the scalability of real time decoders that are currently available. Note that we also view the link from the control system to a window and the one from a window to the runtime as parts of an inner decoder.

An inner decoder's scalability bottleneck depends on its design. As examples, we look at two FPGA-based real time decoders that work for at least medium-sized codes and have different scalability challenges. Helios [79] is a distributed version of a Union-Find decoder that exploits parallel computing resources for speedup. It has been demonstrated on a 21×21 patch and operates with $O(d^3)$ processing elements, each assigned to a node in the decoding graph and communicates via shared memory. Its average runtime per measurement round decreases with d, as long as d is not large enough to require board to board communication between FPGAs (which is much slower than shared memory based communication). Aside from this limit, the main scalability challenge of Helios is the $O(d^3)$ scaling of its computing resources. A recent update of Helios [49] improved its scalability to a 51×51 patch at the cost of increased latency.

Riverlane's Collision Clustering (CC) decoder [11] is another UF-based decoder, with demonstrated implementations on both FPGA and ASIC. The advantage of CC is its efficient use of storage resources, so its scalability is not constrained by hardware capacity, but instead constrained by the throughput. On an FPGA, its average execution time per measurement round increases with d, but is below 1 μs until $d = 23$. Like Helios, each instance of the CC

decoder also needs to fit on 1 FPGA due to the usage of shared memory.

The maximum window size supported by the FPGA is constrained by the scalability of the inner decoder. The decoding graphs in [11, 79] are cubes, but for the spatially parallel windows, the number of measurement rounds that need to be processed at once does not depend on the full width of the window, but rather the width of the commit region. This would decrease the volume of the decoding graph by roughly half, but in practice, applying sliding windows along the time dimension would increase the volume of the decoding graph by $2X$. Therefore we expect that volume of the decoding graph in a window would be similar to the ones used in [11, 79], which means an inner decoder for $d = 51$ translates to a commit region width of ~ 25 for the spatially parallel windows, assuming that the buffer is slightly wider than half of the commit region. We conclude that FPGA-based implementations of real time decoders connected into spatially parallel windows will be able to support at least medium term demonstrations of logical operations. In the long term, ASIC-based implementations should be able to support larger code distances, because they can be optimized for higher performance and lower cost [11].

How does the maximum window size depend on the physical noise p? Recall from Section 4.5 that when p is lower, a wider buffer is required for the spatially parallel windows to achieve good accuracy. Wider buffers lead to larger windows, which impose higher requirements on the inner decoder. But it is worth noting that most decoders also run faster at low p [56], which is natural since physical errors are sparse at low p. If the main constraint on the scalability of an inner decoder is its throughput, as in the case of the CC decoder, then these are competing factors that determine the maximum window size it can support at low p.

4.7 Conclusion

Spatially parallel windows make real time decoding of QEC codes scalable for the large patches of codes that arise during fault-tolerant quantum logical operations. We show that the scheme is compatible with the constraints imposed by using hardware accelerators, with proper window configuration. We perform numerical studies of the decoding accuracy and investigate the mechanisms through which the spatially parallel windows might introduce additional logical errors. The implications of the results impose constraints on mapping and introduce requirements on the window size and buffer width. We specifically study how the requirement on buffer width depends on the physical noise level. We assess the throughput of the decoding scheme and analyze the maximum window size that can be supported before the throughput falls below the requirement. We find that the communication between neighboring windows never becomes the bottleneck in the decoding scheme, so the maximum window size is limited by the scalability of the inner decoders. Since running on larger windows decreases the throughput of the inner decoders and/or requires more hardware resources, the window size and the buffer width need to be carefully chosen to achieve a balance between throughput/cost and accuracy.

CHAPTER 5

CONCLUSION AND OUTLOOK

The main theme of this dissertation is that software and architecture design for quantum computing need to adapt to features and variations of both quantum hardware and algorithms. Research in this direction starts with identifying gaps between the demands of quantum algorithms and the limitations of current quantum devices. These gaps can be closed by developing better quantum hardware or directly optimizing the algorithms, but sometimes it could be more practical to address them with systems-level solutions.

Chapter 2 identifies the problem that two-qubit gates on some superconducting qubits are susceptible to systematic deviations. One way to solve the problem is to reduce the deviations, at the cost of potentially making lower-fidelity gates. We instead propose a framework to accommodate the hardware deviations by choosing the most efficient basis gates from the deviated trajectories. However, two-qubit basis gates are not the only design choice that should be made according to hardware variations. For devices that consist of qubits with different properties, it is important that the compiler is optimized based on these differences. The choices of hardware gates and compiling strategy also highly depend on the type of hardware. Current superconducting devices only support nearest-neighbor connectivity, and interactions that directly act on one or two qubits, so there has been research that aims to reduce the number of SWAP gates in the compiling process. On the neutral atoms, multi-qubit interactions and atom movements are available, but it is challenging to locally access the atoms, and the qubit measurements have a destructive nature. For trapped ions, it is easier to interact ions within the same trap than those in different traps, and it is important to design the structure of the traps. Therefore, these hardware should be supported by different systems.

The same logic also applies to quantum error correction. The decoders for QEC codes should account for the variations on a device, capturing the noise landscape as accurately as

possible, in order to improve the accuracy of decoding. And as explored in Chapter 3, the presence of permanently defective qubits requires adapting QEC codes and modifying the architecture design, exploiting the flexibility of modular architectures. Aside from permanent defects, temporary defects from erasure errors or cosmic rays can also affect QEC.

Defective qubits are only one possible cause of the mismatch between device connectivity and the connectivity requirement of a QEC code. There has also been research that designs QEC codes specifically for devices with limited connectivity, such as the hexagonal lattice. Currently, qLDPC codes are a promising research direction because of their much better encoding rates than the surface code. The long range interactions in these codes require careful mapping from the codes to the devices. It is also a challenge to efficiently implement logical gates on these codes. Heterogeneity might be necessary in future devices that support qLDPC codes. With superconducting qubits, the device might need to include a mix of short-range and long-range connections. With neutral atoms, it might need to use multiple atom species, and a combination of different technologies for addressing the atoms. There has also been proposals to implement logical operations on qLDPC codes by using multiple QEC codes for the same computation, with the surface code as compute qubits. Another promising solution to the high qubit overhead of topological codes is to use bosonic QEC schemes, or a combination of continuous-variable and discrete-variable error correction.

Aside from designing for the hardware features and variations, another research direction is to accommodate the demands of algorithms. Chapter 4 is an attempt towards addressing the problem that real time decoding schemes often do not scale well for the large patches that arise during lattice surgery operations. One can also focus on a particular application, or component of computation, and make optimizations specific to it. For example, magic states are one important component to QEC, so it is important to efficiently produce them. Alternatively, one can identify common bottlenecks of multiple algorithms.

In the future, more advanced quantum computers will be made and new algorithms will

be developed. The field will face different challenges than we do now. However, as we go from abstract algorithms to concrete implementation, it is always important to design proper software and systems that address practical concerns.

REFERENCES

[1] Qiskit Ignis. https://github.com/Qiskit/qiskit-ignis, 2019.

[2] ANSYS Maxwell. https://www.ansys.com/, 2022.

[3] AWR Microwave Office. https://www.awr.com/, 2022.

[4] pyGSTi. https://www.pygsti.info, 2022.

[5] Suppressing quantum errors by scaling a surface code logical qubit. *Nature*, 614(7949):676–681, 2023.

[6] D. M. Abrams, N. Didier, B. R. Johnson, M. P. da Silva, and C. A. Ryan. Implementation of XY entangling gates with a single calibrated pulse. *Nature Electronics*, 3(12):744–750, Nov. 2020.

[7] R. Acharya, I. Aleiner, R. Allen, T. I. Andersen, M. Ansmann, F. Arute, K. Arya, A. Asfaw, J. Atalaya, R. Babbush, D. Bacon, J. C. Bardin, J. Basso, A. Bengtsson, S. Boixo, G. Bortoli, A. Bourassa, J. Bovaird, L. Brill, M. Broughton, B. B. Buckley, D. A. Buell, T. Burger, B. Burkett, N. Bushnell, Y. Chen, Z. Chen, B. Chiaro, J. Cogan, R. Collins, P. Conner, W. Courtney, A. L. Crook, B. Curtin, D. M. Debroy, A. Del Toro Barba, S. Demura, A. Dunsworth, D. Eppens, C. Erickson, L. Faoro, E. Farhi, R. Fatemi, L. Flores Burgos, E. Forati, A. G. Fowler, B. Foxen, W. Giang, C. Gidney, D. Gilboa, M. Giustina, A. Grajales Dau, J. A. Gross, S. Habegger, M. C. Hamilton, M. P. Harrigan, S. D. Harrington, O. Higgott, J. Hilton, M. Hoffmann, S. Hong, T. Huang, A. Huff, W. J. Huggins, L. B. Ioffe, S. V. Isakov, J. Iveland, E. Jeffrey, Z. Jiang, C. Jones, P. Juhas, D. Kafri, K. Kechedzhi, J. Kelly, T. Khattar, M. Khezri, M. Kieferová, S. Kim, A. Kitaev, P. V. Klimov, A. R. Klots, A. N. Korotkov, F. Kostritsa, J. M. Kreikebaum, D. Landhuis, P. Laptev, K.-M. Lau, L. Laws, J. Lee, K. Lee, B. J. Lester, A. Lill, W. Liu, A. Locharla, E. Lucero, F. D. Malone, J. Marshall, O. Martin, J. R. McClean, T. McCourt, M. McEwen, A. Megrant, B. Meurer Costa, X. Mi, K. C. Miao, M. Mohseni, S. Montazeri, A. Morvan, E. Mount, W. Mruczkiewicz, O. Naaman, M. Neeley, C. Neill, A. Nersisyan, H. Neven, M. Newman, J. H. Ng, A. Nguyen, M. Nguyen, M. Y. Niu, T. E. O'Brien, A. Opremcak, J. Platt, A. Petukhov, R. Potter, L. P. Pryadko, C. Quintana, P. Roushan, N. C. Rubin, N. Saei, D. Sank, K. Sankaragomathi, K. J. Satzinger, H. F. Schurkus, C. Schuster, M. J. Shearn, A. Shorter, V. Shvarts, J. Skruzny, V. Smelyanskiy, W. C. Smith, G. Sterling, D. Strain, M. Szalay, A. Torres, G. Vidal, B. Villalonga, C. Vollgraff Heidweiller, T. White, C. Xing, Z. J. Yao, P. Yeh, J. Yoo, G. Young, A. Zalcman, Y. Zhang, N. Zhu, and G. Q. AI. Suppressing quantum errors by scaling a surface code logical qubit. *Nature*, 614(7949):676–681, 2023.

[8] M. S. ANIS, Abby-Mitchell, H. Abraham, AduOffei, R. Agarwal, G. Agliardi, M. Aharoni, I. Y. Akhalwaya, G. Aleksandrowicz, T. Alexander, M. Amy, S. Anagolum, Anthony-Gandon, E. Arbel, A. Asfaw, A. Athalye, A. Avkhadiev, C. Azaustre,

P. BHOLE, A. Banerjee, S. Banerjee, W. Bang, A. Bansal, P. Barkoutsos, A. Barnawal, G. Barron, G. S. Barron, L. Bello, Y. Ben-Haim, M. C. Bennett, D. Bevenius, D. Bhatnagar, A. Bhobe, P. Bianchini, L. S. Bishop, C. Blank, S. Bolos, S. Bopardikar, S. Bosch, S. Brandhofer, Brandon, S. Bravyi, N. Bronn, Bryce-Fuller, D. Bucher, A. Burov, F. Cabrera, P. Calpin, L. Capelluto, J. Carballo, G. Carrascal, A. Carriker, I. Carvalho, A. Chen, C.-F. Chen, E. Chen, J. C. Chen, R. Chen, F. Chevallier, K. Chinda, R. Cholarajan, J. M. Chow, S. Churchill, CisterMoke, C. Claus, C. Clauss, C. Clothier, R. Cocking, R. Cocuzzo, J. Connor, F. Correa, Z. Crockett, A. J. Cross, A. W. Cross, S. Cross, J. Cruz-Benito, C. Culver, A. D. Córcoles-Gonzales, N. D, S. Dague, T. E. Dandachi, A. N. Dangwal, J. Daniel, M. Daniels, M. Dartiailh, A. R. Davila, F. Debouni, A. Dekusar, A. Deshmukh, M. Deshpande, D. Ding, J. Doi, E. M. Dow, E. Drechsler, E. Dumitrescu, K. Dumon, I. Duran, K. EL-Safty, E. Eastman, G. Eberle, A. Ebrahimi, P. Eendebak, D. Egger, ElePT, Emilio, A. Espiricueta, M. Everitt, D. Facoetti, Farida, P. M. Fernández, S. Ferracin, D. Ferrari, A. H. Ferrera, R. Fouilland, A. Frisch, A. Fuhrer, B. Fuller, M. GEORGE, J. Gacon, B. G. Gago, C. Gambella, J. M. Gambetta, A. Gammanpila, L. Garcia, T. Garg, S. Garion, J. R. Garrison, J. Garrison, T. Gates, L. Gil, A. Gilliam, A. Giridharan, J. Gomez-Mosquera, Gonzalo, S. de la Puente González, J. Gorzinski, I. Gould, D. Greenberg, D. Grinko, W. Guan, D. Guijo, J. A. Gunnels, H. Gupta, N. Gupta, J. M. Günther, M. Haglund, I. Haide, I. Hamamura, O. C. Hamido, F. Harkins, K. Hartman, A. Hasan, V. Havlicek, J. Hellmers, Ł. Herok, S. Hillmich, H. Horii, C. Howington, S. Hu, W. Hu, J. Huang, R. Huisman, H. Imai, T. Imamichi, K. Ishizaki, Ishwor, R. Iten, T. Itoko, A. Ivrii, A. Javadi, A. Javadi-Abhari, W. Javed, Q. Jianhua, M. Jivrajani, K. Johns, S. Johnstun, Jonathan-Shoemaker, JosDenmark, JoshDumo, J. Judge, T. Kachmann, A. Kale, N. Kanazawa, J. Kane, Kang-Bae, A. Kapila, A. Karazeev, P. Kassebaum, T. Kehrer, J. Kelso, S. Kelso, V. Khanderao, S. King, Y. Kobayashi, Kovi11Day, A. Kovyrshin, R. Krishnakumar, V. Krishnan, K. Krsulich, P. Kumkar, G. Kus, R. LaRose, E. Lacal, R. Lambert, H. Landa, J. Lapeyre, J. Latone, S. Lawrence, C. Lee, G. Li, J. Lishman, D. Liu, P. Liu, Lolcroc, A. K. M, L. Madden, Y. Maeng, S. Maheshkar, K. Majmudar, A. Malyshev, M. E. Mandouh, J. Manela, Manjula, J. Marecek, M. Marques, K. Marwaha, D. Maslov, P. Maszota, D. Mathews, A. Matsuo, F. Mazhandu, D. McClure, M. McElaney, C. McGarry, D. McKay, D. McPherson, S. Meesala, D. Meirom, C. Mendell, T. Metcalfe, M. Mevissen, A. Meyer, A. Mezzacapo, R. Midha, D. Miller, Z. Minev, A. Mitchell, N. Moll, A. Montanez, G. Monteiro, M. D. Mooring, R. Morales, N. Moran, D. Morcuende, S. Mostafa, M. Motta, R. Moyard, P. Murali, J. Müggenburg, T. NEMOZ, D. Nadlinger, K. Nakanishi, G. Nannicini, P. Nation, E. Navarro, Y. Naveh, S. W. Neagle, P. Neuweiler, A. Ngoueya, T. Nguyen, J. Nicander, Nick-Singstock, P. Niroula, H. Norlen, NuoWenLei, L. J. O'Riordan, O. Ogunbayo, P. Ollitrault, T. Onodera, R. Otaolea, S. Oud, D. Padilha, H. Paik, S. Pal, Y. Pang, A. Panigrahi, V. R. Pascuzzi, S. Perriello, E. Peterson, A. Phan, K. Pilch, F. Piro, M. Pistoia, C. Piveteau, J. Plewa, P. Pocreau, A. Pozas-Kerstjens, R. Pracht, M. Prokop, V. Prutyanov, S. Puri, D. Puzzuoli, J. Pérez, Quant02, Quintiii, R. I. Rahman, A. Raja, R. Rajeev, I. Rajput, N. Ramagiri, A. Rao, R. Raymond,

O. Reardon-Smith, R. M.-C. Redondo, M. Reuter, J. Rice, M. Riedemann, Rietesh, D. Risinger, M. L. Rocca, D. M. Rodríguez, RohithKarur, B. Rosand, M. Rossmannek, M. Ryu, T. SAPV, N. R. C. Sa, A. Saha, A. Ash-Saki, S. Sanand, M. Sandberg, H. Sandesara, R. Sapra, H. Sargsyan, A. Sarkar, N. Sathaye, B. Schmitt, C. Schnabel, Z. Schoenfeld, T. L. Scholten, E. Schoute, M. Schulterbrandt, J. Schwarm, J. Seaward, Sergi, I. F. Sertage, K. Setia, F. Shah, N. Shammah, R. Sharma, Y. Shi, J. Shoemaker, A. Silva, A. Simonetto, D. Singh, D. Singh, P. Singh, P. Singkanipa, Y. Siraichi, Siri, J. Sistos, I. Sitdikov, S. Sivarajah, M. B. Sletfjerding, J. A. Smolin, M. Soeken, I. O. Sokolov, I. Sokolov, V. P. Soloviev, SooluThomas, Starfish, D. Steenken, M. Stypulkoski, A. Suau, S. Sun, K. J. Sung, M. Suwama, O. Słowik, H. Takahashi, T. Takawale, I. Tavernelli, C. Taylor, P. Taylour, S. Thomas, K. Tian, M. Tillet, M. Tod, M. Tomasik, C. Tornow, E. de la Torre, J. L. S. Toural, K. Trabing, M. Treinish, D. Trenev, TrishaPe, F. Truger, G. Tsilimigkounakis, D. Tulsi, W. Turner, Y. Vaknin, C. R. Valcarce, F. Varchon, A. Vartak, A. C. Vazquez, P. Vijaywargiya, V. Villar, B. Vishnu, D. Vogt-Lee, C. Vuillot, J. Weaver, J. Weidenfeller, R. Wieczorek, J. A. Wildstrom, J. Wilson, E. Winston, WinterSoldier, J. J. Woehr, S. Woerner, R. Woo, C. J. Wood, R. Wood, S. Wood, J. Wootton, M. Wright, L. Xing, J. YU, B. Yang, U. Yang, J. Yao, D. Yeralin, R. Yonekura, D. Yonge-Mallo, R. Yoshida, R. Young, J. Yu, L. Yu, C. Zachow, L. Zdanski, H. Zhang, I. Zidaru, C. Zoufal, aeddins ibm, alexzhang13, b63, bartek bartlomiej, bcamorrison, brandhsn, charmerDark, deeplokhande, dekel.meirom, dime10, dlasecki, ehchen, fanizzamarco, fs1132429, gadial, galeinston, georgezhou20, georgios ts, gruu, hhorii, hykavitha, itoko, jeppevinkel, jessica angel7, jezerjojo14, jliu45, jscott2, klinvill, krutik2966, ma5x, michelle4654, msuwama, nico lgrs, ntgiwsvp, ordmoj, sagar pahwa, pritamsinha2304, ryancocuzzo, saktar unr, saswati qiskit, septembrr, sethmerkel, sg495, shaashwat, smturro2, sternparky, strickroman, tigerjack, tsura crisaldo, upsideon, vadebayo49, welien, willhbang, wmurphy collabstar, yang.luh, and M. Čepulkovskis. Qiskit: An open-source framework for quantum computing, 2021.

[9] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. S. L. Brandao, D. A. Buell, B. Burkett, Y. Chen, Z. Chen, B. Chiaro, R. Collins, W. Courtney, A. Dunsworth, E. Farhi, B. Foxen, A. Fowler, C. Gidney, M. Giustina, R. Graff, K. Guerin, S. Habegger, M. P. Harrigan, M. J. Hartmann, A. Ho, M. Hoffmann, T. Huang, T. S. Humble, S. V. Isakov, E. Jeffrey, Z. Jiang, D. Kafri, K. Kechedzhi, J. Kelly, P. V. Klimov, S. Knysh, A. Korotkov, F. Kostritsa, D. Landhuis, M. Lindmark, E. Lucero, D. Lyakh, S. Mandrà, J. R. McClean, M. McEwen, A. Megrant, X. Mi, K. Michielsen, M. Mohseni, J. Mutus, O. Naaman, M. Neeley, C. Neill, M. Y. Niu, E. Ostby, A. Petukhov, J. C. Platt, C. Quintana, E. G. Rieffel, P. Roushan, N. C. Rubin, D. Sank, K. J. Satzinger, V. Smelyanskiy, K. J. Sung, M. D. Trevithick, A. Vainsencher, B. Villalonga, T. White, Z. J. Yao, P. Yeh, A. Zalcman, H. Neven, and J. M. Martinis. Quantum supremacy using a programmable superconducting processor. *Nature*, 574:505–510, 2019.

[10] J. M. Auger, H. Anwar, M. Gimeno-Segovia, T. M. Stace, and D. E. Browne. Fault-

tolerance thresholds for the surface code with fabrication errors. *Physical Review A*, 96(4):042316, 2017.

[11] B. Barber, K. M. Barnes, T. Bialas, O. Buğdaycı, E. T. Campbell, N. I. Gillespie, K. Johar, R. Rajan, A. W. Richardson, L. Skoric, et al. A real-time, scalable, fast and highly resource efficient decoder for a quantum computer. *arXiv preprint arXiv:2309.05558*, 2023.

[12] E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM Journal on computing*, 26(5):1411–1473, 1997.

[13] M. E. Beverland, B. J. Brown, M. J. Kastoryano, and Q. Marolleau. The role of entropy in topological quantum error correction. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(7):073404, 2019.

[14] R. Blume-Kohout, M. P. da Silva, E. Nielsen, T. Proctor, K. Rudinger, M. Sarovar, and K. Young. A taxonomy of small markovian errors. *PRX Quantum*, 3(2), may 2022.

[15] H. Bombín, C. Dawson, Y.-H. Liu, N. Nickerson, F. Pastawski, and S. Roberts. Modular decoding: parallelizable real-time decoding for quantum computers. *arXiv preprint arXiv:2303.04846*, 2023.

[16] H. Bombin and M. A. Martin-Delgado. Topological quantum error correction with optimal encoding rate. *Phys. Rev. A*, 73:062303, Jun 2006.

[17] S. Bravyi and A. Kitaev. Universal quantum computation with ideal clifford gates and noisy ancillas. *Phys. Rev. A*, 71:022316, Feb 2005.

[18] S. Bravyi and A. Vargo. Simulation of rare events in quantum error correction. *Phys. Rev. A*, 88:062308, Dec 2013.

[19] S. Bravyi and A. Vargo. Simulation of rare events in quantum error correction. *Physical Review A*, 88(6):062308, 2013.

[20] M. J. Bremner, C. M. Dawson, J. L. Dodd, A. Gilchrist, A. W. Harrow, D. Mortimer, M. A. Nielsen, and T. J. Osborne. Practical scheme for quantum computation with any two-qubit entangling gate. *Physical review letters*, 89(24):247902, 2002.

[21] B. J. Brown. Conservation laws and quantum error correction: Towards a generalised matching decoder. *IEEE BITS the Information Theory Magazine*, pages 1–12, 2023.

[22] B. J. Brown, K. Laubscher, M. S. Kesselring, and J. R. Wootton. Poking holes and cutting corners to achieve clifford gates with the surface code. *Phys. Rev. X*, 7:021029, May 2017.

[23] C. Chamberland and E. T. Campbell. Circuit-level protocol and analysis for twist-based lattice surgery. *Physical Review Research*, 4(2):023090, 2022.

[24] C. Chamberland and E. T. Campbell. Universal quantum computing with twist-free and temporally encoded lattice surgery. *PRX Quantum*, 3(1):010331, 2022.

[25] C. Chamberland, L. Goncalves, P. Sivarajah, E. Peterson, and S. Grimberg. Techniques for combining fast local decoders with global decoders under circuit-level noise. *Quantum Science and Technology*, 8(4):045011, 2023.

[26] I. L. Chuang and M. A. Nielsen. Prescription for experimental determination of the dynamics of a quantum black box. *Journal of Modern Optics*, 44(11-12):2455–2467, 1997.

[27] G. E. Crooks. Gates, states, and circuits. 2020.

[28] S. A. Cuccaro, T. G. Draper, S. A. Kutin, and D. P. Moulton. A new quantum ripple-carry addition circuit. *arXiv preprint quant-ph/0410184*, 2004.

[29] P. Das, A. Locharla, and C. Jones. Lilliput: a lightweight low-latency lookup-table decoder for near-term quantum error correction. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 541–553, 2022.

[30] P. Das, C. A. Pattison, S. Manne, D. M. Carmean, K. M. Svore, M. Qureshi, and N. Delfosse. Afs: Accurate, fast, and scalable error-decoding for fault-tolerant quantum computers. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 259–273. IEEE, 2022.

[31] N. Delfosse, A. Paz, A. Vaschillo, and K. M. Svore. How to choose a decoder for a fault-tolerant quantum computer? the speed vs accuracy trade-off. *arXiv preprint arXiv:2310.15313*, 2023.

[32] N. Delfosse and G. Zémor. Linear-time maximum likelihood decoding of surface codes over the quantum erasure channel. *Physical Review Research*, 2(3):033042, 2020.

[33] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill. Topological quantum memory. *Journal of Mathematical Physics*, 43(9):4452–4505, 2002.

[34] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill. Topological quantum memory. *Journal of Mathematical Physics*, 43(9):4452–4505, 2002.

[35] C. Developers. Cirq. Aug 2021. See full list of authors on Github: https://github.com/quantumlib/Cirq/graphs/contributors.

[36] J. Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of research of the National Bureau of Standards B*, 69(125-130):55–56, 1965.

[37] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965.

[38] E. Farhi, J. Goldstone, and S. Gutmann. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028*, 2014.

[39] A. G. Fowler, S. J. Devitt, and C. Jones. Surface code implementation of block code state distillation. *Scientific reports*, 3(1):1939, 2013.

[40] A. G. Fowler and C. Gidney. Low overhead quantum computation using lattice surgery. *arXiv preprint arXiv:1808.06709*, 2018.

[41] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86(3):032324, 2012.

[42] B. Foxen, C. Neill, A. Dunsworth, P. Roushan, B. Chiaro, A. Megrant, J. Kelly, Z. Chen, K. Satzinger, R. Barends, F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, S. Boixo, D. Buell, B. Burkett, Y. Chen, R. Collins, E. Farhi, A. Fowler, C. Gidney, M. Giustina, R. Graff, M. Harrigan, T. Huang, S. V. Isakov, E. Jeffrey, Z. Jiang, D. Kafri, K. Kechedzhi, P. Klimov, A. Korotkov, F. Kostritsa, D. Landhuis, E. Lucero, J. McClean, M. McEwen, X. Mi, M. Mohseni, J. Y. Mutus, O. Naaman, M. Neeley, M. Niu, A. Petukhov, C. Quintana, N. Rubin, D. Sank, V. Smelyanskiy, A. Vainsencher, T. C. White, Z. Yao, P. Yeh, A. Zalcman, H. Neven, and J. M. Martinis. Demonstrating a continuous set of two-qubit gates for near-term quantum algorithms. *Phys. Rev. Lett.*, 125:120504, Sep 2020.

[43] F. N. M. Froning, L. C. Camenzind, O. A. H. van der Molen, A. Li, E. P. A. M. Bakkers, D. M. Zumbühl, and F. R. Braakman. Ultrafast hole spin qubit with gate-tunable spin–orbit switch functionality. *Nature Nanotechnology*, 16(3):308–312, 2021.

[44] J. M. Gambetta, J. M. Chow, and M. Steffen. Building logical qubits in a superconducting quantum computing system. *npj Quantum Information*, 3(1):1–7, 2017.

[45] C. Gidney. Stim: a fast stabilizer circuit simulator. *Quantum*, 5:497, July 2021.

[46] C. Gidney. Stability Experiments: The Overlooked Dual of Memory Experiments. *Quantum*, 6:786, Aug. 2022.

[47] C. Gidney and M. Ekerå. How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits. *Quantum*, 5:433, 2021.

[48] C. Gidney and A. G. Fowler. Efficient magic state factories with a catalyzed $|CCZ\rangle$ to $2|T\rangle$ transformation. *Quantum*, 3:135, Apr. 2019.

[49] N. Godawatte Liyanage, Y. Wu, S. Tagare, and L. Zhong. Multi-fpga union-find decoder for surface codes. *Bulletin of the American Physical Society*, 2024.

[50] D. Greenbaum. Introduction to quantum gate set tomography. *arXiv preprint arXiv:1509.02921*, 2015.

[51] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[52] C. Guinn, S. Sussman, P. S. Mundada, A. Vrajitoarea, C. Leroux, A. Place, C. L. Calonnec, A. Di Paolo, A. Petrescu, A. Blais, and A. A. Houck. Fast parametrically driven entangling gates in superconducting circuits using a tunable coupler. *Bulletin of the American Physical Society*, 2022.

[53] N. W. Hendrickx, W. I. L. Lawrie, M. Russ, F. van Riggelen, S. L. de Snoo, R. N. Schouten, A. Sammak, G. Scappucci, and M. Veldhorst. A four-qubit germanium quantum processor. *Nature*, 591(7851):580–585, 2021.

[54] J. B. Hertzberg, E. J. Zhang, S. Rosenblatt, E. Magesan, J. A. Smolin, J.-B. Yau, V. P. Adiga, M. Sandberg, M. Brink, J. M. Chow, et al. Laser-annealing josephson junctions for yielding scaled-up superconducting quantum processors. *npj Quantum Information*, 7(1):1–8, 2021.

[55] J. B. Hertzberg, E. J. Zhang, S. Rosenblatt, E. Magesan, J. A. Smolin, J.-B. Yau, V. P. Adiga, M. Sandberg, M. Brink, J. M. Chow, et al. Laser-annealing josephson junctions for yielding scaled-up superconducting quantum processors. *npj Quantum Information*, 7(1):1–8, 2021.

[56] O. Higgott and C. Gidney. Sparse blossom: correcting a million errors per core second with minimum-weight matching. *arXiv preprint arXiv:2303.15933*, 2023.

[57] A. D. Hill, M. J. Hodson, N. Didier, and M. J. Reagor. Realization of arbitrary doubly-controlled quantum phase gates, 2021.

[58] A. Holmes, M. R. Jokar, G. Pasandi, Y. Ding, M. Pedram, and F. T. Chong. Nisq+: Boosting quantum computing power by approximating quantum error correction. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 556–569. IEEE, 2020.

[59] C. Horsman, A. G. Fowler, S. Devitt, and R. V. Meter. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14(12):123011, dec 2012.

[60] C. Horsman, A. G. Fowler, S. Devitt, and R. Van Meter. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14(12):123011, 2012.

[61] C. Huang, D. Ding, F. Wu, L. Kong, F. Zhang, X. Ni, Y. Shi, H.-h. Zhao, and J. Chen. Towards ultra-high fidelity quantum operations: Sqisw gate as a native two-qubit gate. *arXiv preprint arXiv:2105.06074*, 2021.

[62] A. R. Iyengar, M. Papaleo, P. H. Siegel, J. K. Wolf, A. Vanelli-Coralli, and G. E. Corazza. Windowed decoding of protograph-based ldpc convolutional codes over erasure channels. *IEEE Transactions on Information Theory*, 58(4):2303–2320, 2011.

[63] P. Jurcevic, A. Javadi-Abhari, L. S. Bishop, I. Lauer, D. F. Bogorin, M. Brink, L. Capelluto, O. Günlük, T. Itoko, N. Kanazawa, et al. Demonstration of quantum volume 64 on a superconducting quantum computing system. *Quantum Science and Technology*, 6(2):025020, 2021.

[64] A. Kandala, K. X. Wei, S. Srinivasan, E. Magesan, S. Carnevale, G. A. Keefe, D. Klaus, O. Dial, and D. C. McKay. Demonstration of a high-fidelity cnot for fixed-frequency transmons with engineered zz suppression, 2020.

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

[66] E. Knill, D. Leibfried, R. Reichle, J. Britton, R. B. Blakestad, J. D. Jost, C. Langer, R. Ozeri, S. Seidelin, and D. J. Wineland. Randomized benchmarking of quantum gates. *Physical Review A*, 77(1):012307, 2008.

[67] J. Kreikebaum, K. O'Brien, A. Morvan, and I. Siddiqi. Improving wafer-scale josephson junction resistance variation in superconducting quantum coherent circuits. *Superconductor Science and Technology*, 33(6):06LT02, 2020.

[68] J. M. e. a. Kreikebaum. Improving wafer-scale josephson junction resistance variation in superconducting quantum coherent circuits. *Supercond. Sci. Technol.*, 33:033201, 2020.

[69] S. Krinner, N. Lacroix, A. Remm, A. Di Paolo, E. Genois, C. Leroux, C. Hellings, S. Lazar, F. Swiadek, J. Herrmann, et al. Realizing repeated quantum error correction in a distance-three surface code. *Nature*, 605(7911):669–674, 2022.

[70] S. Krinner, N. Lacroix, A. Remm, A. Di Paolo, E. Genois, C. Leroux, C. Hellings, S. Lazar, F. Swiadek, J. Herrmann, G. J. Norris, C. K. Andersen, M. Müller, A. Blais, C. Eichler, and A. Wallraff. Realizing repeated quantum error correction in a distance-three surface code. *Nature*, 605(7911):669–674, 2022.

[71] J. Ku, X. Xu, M. Brink, D. C. McKay, J. B. Hertzberg, M. H. Ansari, and B. L. T. Plourde. Suppression of unwanted zz interactions in a hybrid two-qubit system. *Phys. Rev. Lett.*, 125:200504, Nov 2020.

[72] S. Kwon, A. Tomonaga, G. L. Bhai, S. J. Devitt, and J.-S. Tsai. Gate-based superconducting quantum computing. *Journal of Applied Physics*, 129(4):041102, Jan. 2021.

[73] L. Lao, A. Korotkov, Z. Jiang, W. Mruczkiewicz, T. E. O'Brien, and D. E. Browne. Software mitigation of coherent two-qubit gate errors. *Quantum Science and Technology*, 7(2):025021, 2022.

[74] L. Lao, P. Murali, M. Martonosi, and D. Browne. Designing calibration and expressivity-efficient instruction sets for quantum computing. In *2021 ACM/IEEE*

48th Annual International Symposium on Computer Architecture (ISCA), pages 846–859. IEEE, 2021.

[75] G. Li, Y. Ding, and Y. 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, 2019.

[76] D. Litinski. A Game of Surface Codes: Large-Scale Quantum Computing with Lattice Surgery. *Quantum*, 3:128, Mar. 2019.

[77] D. Litinski. A game of surface codes: Large-scale quantum computing with lattice surgery. *Quantum*, 3:128, 2019.

[78] D. Litinski and F. von Oppen. Lattice surgery with a twist: simplifying clifford gates of surface codes. *Quantum*, 2:62, 2018.

[79] N. Liyanage, Y. Wu, A. Deters, and L. Zhong. Scalable quantum error correction for surface codes using fpga. *arXiv preprint arXiv:2301.08419*, 2023.

[80] M. T. Mądzik, S. Asaad, A. Youssry, B. Joecker, K. M. Rudinger, E. Nielsen, K. C. Young, T. J. Proctor, A. D. Baczewski, A. Laucht, V. Schmitt, F. E. Hudson, K. M. Itoh, A. M. Jakob, B. C. Johnson, D. N. Jamieson, A. S. Dzurak, C. Ferrie, R. Blume-Kohout, and A. Morello. Precision tomography of a three-qubit donor quantum processor in silicon. *Nature*, 601(7893):348–353, 2022.

[81] E. Magesan, J. M. Gambetta, and J. Emerson. Scalable and robust randomized benchmarking of quantum processes. *Physical review letters*, 106(18):180504, 2011.

[82] E. Magesan, J. M. Gambetta, B. R. Johnson, C. A. Ryan, J. M. Chow, S. T. Merkel, M. P. Da Silva, G. A. Keefe, M. B. Rothwell, T. A. Ohki, et al. Efficient measurement of quantum gate error by interleaved randomized benchmarking. *Physical review letters*, 109(8):080505, 2012.

[83] M. McEwen, D. Kafri, Z. Chen, J. Atalaya, K. Satzinger, C. Quintana, P. V. Klimov, D. Sank, C. Gidney, A. Fowler, et al. Removing leakage-induced correlated errors in superconducting quantum error correction. *Nature communications*, 12(1):1761, 2021.

[84] D. C. McKay, S. Filipp, A. Mezzacapo, E. Magesan, J. M. Chow, and J. M. Gambetta. Universal gate for fixed-frequency qubits via a tunable bus. *Phys. Rev. Applied*, 6:064007, Dec 2016.

[85] S. T. Merkel, J. M. Gambetta, J. A. Smolin, S. Poletto, A. D. Córcoles, B. R. Johnson, C. A. Ryan, and M. Steffen. Self-consistent quantum process tomography. *Phys. Rev. A*, 87:062119, 2013.

[86] I. N. Moskalenko, I. A. Simakov, N. N. Abramov, A. A. Grigorev, D. O. Moskalev, A. A. Pishchimova, N. S. Smirnov, E. V. Zikiy, I. A. Rodionov, and I. S. Besedin. High fidelity two-qubit gates on fluxoniums using a tunable coupler, 2022.

[87] C. Müller, J. H. Cole, and J. Lisenfeld. Towards understanding two-level-systems in amorphous solids: insights from quantum circuits. *Reports on Progress in Physics*, 82(12):124501, 2019.

[88] P. Mundada, G. Zhang, T. Hazard, and A. Houck. Suppression of qubit crosstalk in a tunable coupling superconducting circuit. *Phys. Rev. Applied*, 12:054023, Nov 2019.

[89] S. Nagayama, A. G. Fowler, D. Horsman, S. J. Devitt, and R. Van Meter. Surface code error correction on a defective lattice. *New Journal of Physics*, 19(2):023050, 2017.

[90] E. Nielsen, J. K. Gamble, K. Rudinger, T. Scholten, K. Young, and R. Blume-Kohout. Gate set tomography. *Quantum*, 5:557, 2021.

[91] A. Noguchi, A. Osada, S. Masuda, S. Kono, K. Heya, S. P. Wolski, H. Takahashi, T. Sugiyama, D. Lachance-Quirion, and Y. Nakamura. Fast parametric two-qubit gates with suppressed residual interaction using a parity-violated superconducting qubit, 2020.

[92] J. O'Gorman and E. T. Campbell. Quantum computation with realistic magic-state factories. *Physical Review A*, 95(3):032338, 2017.

[93] D. R. Perez, P. Varosy, Z. Li, T. Roy, E. Kapit, and D. Schuster. Error-divisible two-qubit gates, 2021.

[94] E. C. Peterson, L. S. Bishop, and A. Javadi-Abhari. Optimal synthesis into fixed xx interactions. *arXiv preprint arXiv:2111.02535*, 2021.

[95] E. C. Peterson, G. E. Crooks, and R. S. Smith. Two-qubit circuit depth and the monodromy polytope. *Quantum*, 4:247, 2020.

[96] A. Petrescu, C. L. Calonnec, C. Leroux, A. Di Paolo, P. Mundada, S. Sussman, A. Vrajitoarea, A. A. Houck, and A. Blais. Accurate methods for the analysis of strong-drive effects in parametric gates. *arXiv preprint arXiv:2107.02343*, 2021.

[97] A. P. M. Place, L. V. H. Rodgers, P. Mundada, B. M. Smitham, M. Fitzpatrick, Z. Leng, A. Premkumar, J. Bryon, A. Vrajitoarea, S. Sussman, G. Cheng, T. Madhavan, H. K. Babla, X. H. Le, Y. Gang, B. Jäck, A. Gyenis, N. Yao, R. J. Cava, N. P. de Leon, and A. A. Houck. New material platform for superconducting transmon qubits with coherence times exceeding 0.3 milliseconds. *Nature Communications*, 12(1):1779, 2021.

[98] J. Poyatos, J. I. Cirac, and P. Zoller. Complete characterization of a quantum process: the two-bit quantum gate. *Physical Review Letters*, 78(2):390, 1997.

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

[100] G. S. Ravi, J. M. Baker, A. Fayyazi, S. F. Lin, A. Javadi-Abhari, M. Pedram, and F. T. Chong. Better than worst-case decoding for quantum error correction. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 88–102, 2023.

[101] Vivado Design Suite. *Aurora 64B/66B LogiCORE IP Product Guide (PG074)*. AMD.

[102] A. Rezakhani. Characterization of two-qubit perfect entanglers. *Physical Review A*, 70(5):052313, 2004.

[103] L. Ruiz-Perez and J. C. Garcia-Escartin. Quantum arithmetic with the quantum fourier transform. *Quantum Information Processing*, 16(6):1–14, 2017.

[104] C. Ryan, M. Laforest, and R. Laflamme. Randomized benchmarking of single-and multi-qubit control in liquid-state nmr quantum information processing. *New Journal of Physics*, 11(1):013034, 2009.

[105] S. Schlör, J. Lisenfeld, C. Müller, A. Bilmes, A. Schneider, D. P. Pappas, A. V. Ustinov, and M. Weides. Correlating decoherence in transmon qubits: Low frequency noise by single fluctuators. *Phys. Rev. Lett.*, 123:190502, Nov 2019.

[106] V. V. Shende, S. S. Bullock, and I. L. Markov. Recognizing small-circuit structure in two-qubit operators and timing hamiltonians to compute controlled-not gates. *arXiv preprint quant-ph/0308045*, 2003.

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

[108] A. Siegel, A. Strikis, T. Flatters, and S. Benjamin. Adaptive surface code for quantum error correction in the presence of temporary or permanent defects. *arXiv preprint arXiv:2211.08468*, 2022.

[109] L. Skoric, D. E. Browne, K. M. Barnes, N. I. Gillespie, and E. T. Campbell. Parallel window decoding enables scalable fault tolerant quantum computation. *Nature Communications*, 14(1):7040, 2023.

[110] K. N. Smith, G. S. Ravi, J. M. Baker, and F. T. Chong. Scaling superconducting quantum computers with chiplet architectures. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1092–1109. IEEE, 2022.

[111] S. C. Smith, B. J. Brown, and S. D. Bartlett. Local predecoder to reduce the bandwidth and latency of quantum error correction. *Physical Review Applied*, 19(3):034050, 2023.

[112] M. Spiecker, P. Paluch, N. Drucker, S. Matityahu, D. Gusenkova, N. Gosling, S. Günzler, D. Rieger, I. Takmakov, F. Valenti, P. Winkel, R. Gebauer, O. Sander, G. Catelani, A. Shnirman, A. V. Ustinov, W. Wernsdorfer, Y. Cohen, and I. M. Pop. A quantum szilard engine for two-level systems coupled to a qubit, 2022.

[113] T. M. Stace and S. D. Barrett. Error correction and degeneracy in surface codes suffering loss. *Phys. Rev. A*, 81:022317, Feb 2010.

[114] T. M. Stace, S. D. Barrett, and A. C. Doherty. Thresholds for topological codes in the presence of loss. *Phys. Rev. Lett.*, 102:200501, May 2009.

[115] A. Strikis, S. C. Benjamin, and B. J. Brown. Quantum computing is scalable on a planar array of qubits with fabrication defects. *arXiv preprint arXiv:2111.06432*, 2021.

[116] N. Sundaresan, T. J. Yoder, Y. Kim, M. Li, E. H. Chen, G. Harper, T. Thorbeck, A. W. Cross, A. D. Córcoles, and M. Takita. Matching and maximum likelihood decoding of a multi-round subsystem quantum error correction experiment, 2022.

[117] Y. Sung, L. Ding, J. Braumüller, A. Vepsäläinen, B. Kannan, M. Kjaergaard, A. Greene, G. O. Samach, C. McNally, D. Kim, A. Melville, B. M. Niedzielski, M. E. Schwartz, J. L. Yoder, T. P. Orlando, S. Gustavsson, and W. D. Oliver. Realization of high-fidelity cz and zz-free iswap gates with a tunable coupler. *Phys. Rev. X*, 11:021058, Jun 2021.

[118] Y. Suzuki, T. Sugiyama, T. Arai, W. Liao, K. Inoue, and T. Tanimoto. Q3de: A fault-tolerant quantum computer architecture for multi-bit burst errors by cosmic rays. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1110–1125. IEEE, 2022.

[119] X. Tan, F. Zhang, R. Chao, Y. Shi, and J. Chen. Scalable surface code decoders with parallelization in time. *arXiv preprint arXiv:2209.09219*, 2022.

[120] B. M. Terhal. Quantum error correction for quantum memories. *Reviews of Modern Physics*, 87(2):307, 2015.

[121] Y. Tomita and K. M. Svore. Low-distance surface codes under realistic quantum noise. *Phys. Rev. A*, 90:062320, Dec 2014.

[122] Y. Ueno, M. Kondo, M. Tanaka, Y. Suzuki, and Y. Tabuchi. Qecool: On-line quantum error correction with a superconducting decoder for surface code. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 451–456. IEEE, 2021.

[123] S. Vittal, P. Das, and M. Qureshi. Astrea: Accurate quantum error-decoding via practical minimum-weight perfect-matching. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pages 1–16, 2023.

[124] U. Vool and M. Devoret. Introduction to quantum electromagnetic circuits. In *International Journal of Circuit Theory and Applications*, volume 45, 2017.

[125] C. Wang, C. Axline, Y. Y. Gao, T. Brecht, Y. Chu, L. Frunzio, M. H. Devoret, and R. J. Schoelkopf. Surface participation and dielectric loss in superconducting qubits. *Applied Physics Letters*, 107, 2015.

[126] C. Wang, J. Harrington, and J. Preskill. Confniement-higgs transition in a disordered gauge theory and the accuracy threshold for quantum memory. *Ann. Phys.*, 303:31, 2003.

[127] F. H. E. Watson and S. D. Barrett. Logical error rate scaling of the toric code. *New Journal of Physics*, 16(9):093045, sep 2014.

[128] K. X. Wei, E. Magesan, I. Lauer, S. Srinivasan, D. F. Bogorin, S. Carnevale, G. A. Keefe, Y. Kim, D. Klaus, W. Landers, N. Sundaresan, C. Wang, E. J. Zhang, M. Steffen, O. E. Dial, D. C. McKay, and A. Kandala. Hamiltonian engineering with multicolor drives for fast entangling gates and quantum crosstalk cancellation. *Phys. Rev. Lett.*, 129:060501, Aug 2022.

[129] A. Wu, G. Li, H. Zhang, G. G. Guerreschi, Y. Ding, and Y. Xie. A synthesis framework for stitching surface code with superconducting quantum devices. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pages 337–350, 2022.

[130] Y. Wu, N. Liyanage, and L. Zhong. An interpretation of union-find decoder on weighted graphs. *arXiv preprint arXiv:2211.03288*, 2022.

[131] Y. Wu and L. Zhong. Fusion blossom: Fast mwpm decoders for qec. *arXiv preprint arXiv:2305.08307*, 2023.

[132] H. Xiong, Q. Ficheux, A. Somoroff, L. B. Nguyen, E. Dogan, D. Rosenstock, C. Wang, K. N. Nesterov, M. G. Vavilov, and V. E. Manucharyan. Arbitrary controlled-phase gate on fluxonium qubits using differential ac stark shifts. *Phys. Rev. Research*, 4:023040, Apr 2022.

[133] X. Xu and M. H. Ansari. Parasitic-free gate: A protected switch between idle and entangled states, 2022.

[134] X. Xue, M. Russ, N. Samkharadze, B. Undseth, A. Sammak, G. Scappucci, and L. M. K. Vandersypen. Quantum logic with spin qubits crossing the surface code threshold. *Nature*, 601(7893):343–347, 2022.

[135] P. Zanardi, C. Zalka, and L. Faoro. Entangling power of quantum evolutions. *Physical Review A*, 62(3):030301, 2000.

[136] J. Zhang, J. Vala, S. Sastry, and K. B. Whaley. Geometric theory of nonlocal two-qubit operations. *Physical Review A*, 67(4):042313, 2003.

[137] J. Zhang, J. Vala, S. Sastry, and K. B. Whaley. Minimum construction of two-qubit quantum operations. *Physical review letters*, 93(2):020502, 2004.

[138] P. Zhao, D. Lan, P. Xu, G. Xue, M. Blank, X. Tan, H. Yu, and Y. Yu. Suppression of static zz interaction in an all-transmon quantum processor, 2020.

[139] Y. Zhao, Y. Ye, H.-L. Huang, Y. Zhang, D. Wu, H. Guan, Q. Zhu, Z. Wei, T. He, S. Cao, et al. Realization of an error-correcting surface code with superconducting qubits. *Physical Review Letters*, 129(3):030501, 2022.